BATAILLE NAVALE
Tora, Tora, Tora

Grâce au programme Navale, vous allez pouvoir affiner vos qualités de stratège en affrontant votre ST dans une grande bataille navale.

La bataille navale est un jeu de société où s'affrontent deux joueurs, disposant chacun d'une flotte de navires de guerre. Ces derniers sont disposés sur une grille de 10x10 cases. Les flottes sont composées de 4 sous-marins, de 3 torpilleurs, de 2 croiseurs et d'un porte-avions. Plus le navire est grand, plus il occupe de cases. Un porte-avions occupe 4 cases, un croiseur 3 cases, un torpilleur occupe 2 cases et un sous-marin qu'une seule case. Les navires peuvent être placés n'importe où sur l'écran, à condition de respecter une distance d'au moins une case entre chaque bâtiment. Cette règle évite qu'un joueur ne se trompe sur la position d'un navire au cas où deux navires seraient placés l'un contre l'autre.
Le combat est une succession de tirs. Chaque joueur sélectionne une case et tire dessus. Si le tir ne touche aucun navire, l'autre joueur annonce «à l'eau». Dans le cas contraire, il annonce «touché» ou «coulé» si le tir vient de détruire le dernier élément d'un navire. La bataille cesse lorsque l'une des 2 flottes est totalement détruite. Une règle optionnelle que nous n'avons pas retenue est d'annoncer «en vue», lorsque l'un des tirs tombe à une case de distance d'un navire.

Analyse du jeu

Avant d'écrire un programme de jeu, il faut étudier en détail ce que le programme et le joueur doivent faire. Cet analyse définit le fonctionnement du programme, mais surtout les fonctionnalités que le logiciel apporte au joueur. Cet analyse de fonctionnalité est indispensable pour définir l'interface utilisateur. Les principales phases de jeu sont le positionnement de la flotte et le combat.

Positionnement de la flotte par le joueur

Pour que le joueur puisse placer ses navires où il veut, il faut une représentation graphique de la grille de jeu et un système pour sélectionner et placer sur la grille un navire d'un type précis (voir tableau.1).
Cette image vous montre l'aspect de l'interface utilisateur de positionnement de la flotte. Pour placer un navire sur la grille, il suffit de cliquer sur une case. Les options SOUS-MARINS, TORPILLEURS, CROISEURS et PORTE-AVION modifient le type de navire courant. Des cases situées à droite de ces options contiennent le nombre de navires restant à placer. Au lancement du programme, le type de navire par défaut est le sous-marin. L'option courante est affichée sur un fond de couleur rouge, alors que les autres options sont affichées avec un fond de couleur gris. La couleur rouge indique la notion d'énergie et d'activité, tandis que la couleur grise correspond à une non-activité. Grâce à ces couleurs, l'utilisateur doit comprendre intuitivement l'état de l'interface, sans avoir à faire un effort de réflexion consciente.
Les navires peuvent êtres placés horizontalement ou verticalement. Deux options situées en bas de l'écran permettent de préciser le sens de placement des navires. Par défaut, le sens est horizontal. L'option active est affichée sur un fond de couleur violet et l'option inactive sur un fond de couleur jaune. Les couleurs choisies ne sont pas le rouge et

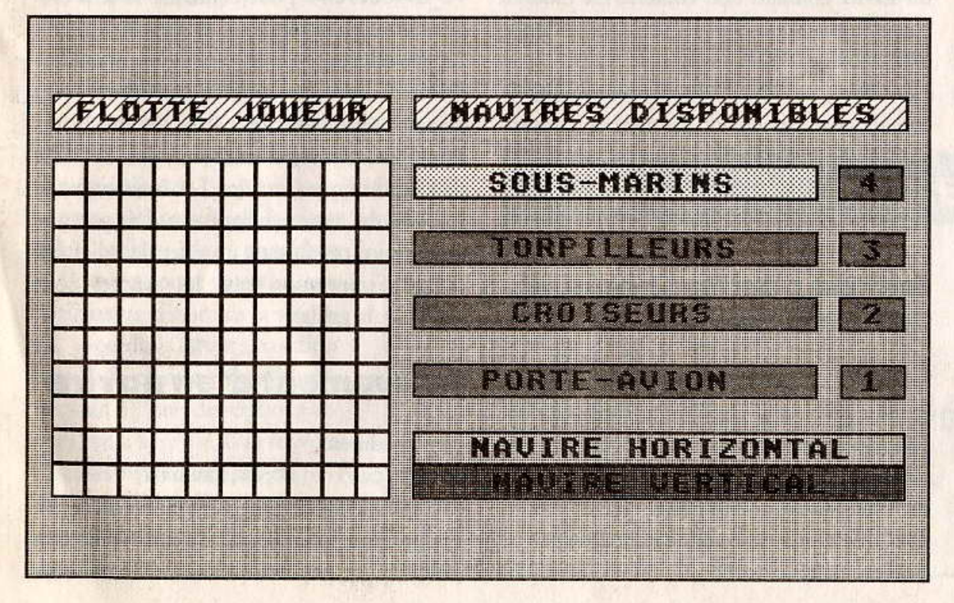


TABLEAU 1

0	0	0	0	1	0	0	0	2	0
0	1	0	0	0	0	0	0	2	0
0	0	0	4	4	4	4	0	0	0
0	0	0	0	0	0	0	0	0	0
0	3	3	3	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	1	0	3	0
2	0	0	2	2	0	0	0	3	0
0	0	0	0	0	0	1	0	3	0
0	0	0	0	0	0	0	0	0	0

0 = pas de navire
1 = Sous-marin
2 = Torpilleur
3 = Croiseur
4 = Porte-avions

le gris afin d'éviter toute confusion visuelle avec les options de sélection de navire.

Quelques tests exécutés avec un premier prototype de l'interface ont montré qu'il fallait retirer un navire de la grille. La solution la plus pratique pour retirer un navire de la grille est de cliquer dessus.

Lorsque le joueur a placé tous les navires de sa flotte, les options `NAVIRE HORIZONTAL` et `NAVIRE VERTICAL` disparaissent et sont remplacées par une option `CONTINUER LE JEU`. Cette option indique au logiciel de passer à la phase de combat. Si le joueur retire un navire de la grille en cliquant dessus, cette option disparaît et est remplacée à nouveau par les options `NAVIRE VERTICAL` et `NAVIRE HORIZONTAL`

Fonctionnement interne

Les problèmes qui se posent pour réaliser le module de placement des navires sont le stockage en mémoire de la flotte du joueur, les tests de positionnement des navires, l'enlèvement d'un navire de la grille et la gestion de l'interface utilisateur.

La présence de 4 types de navires complexifie considérablement la programmation.

Pour la simplifier, au lieu d'utiliser des numéros pour désigner les navires (1,2,3,4), il est possible d'utiliser des constantes numériques. Comme ces dernières n'existent pas en *GFA Basic*, il faut les simuler en utilisant des variables dont le contenu est initialisé au début du programme et ne change pas lors de l'exécution. Ces constantes sont les variables `sous_marin%`, `torpilleur%`, `croiseur%` et `porte_avion%`, utilisées un peu partout dans le programme. C'est une technique que vous gagneriez à reprendre dans vos propres logiciels. Les 16 couleurs vidéo possèdent aussi leurs propres constantes numériques.

```
sous_marin%=1
torpilleur%=2
croiseur%=3
porte_avion%=4
```

Les données de la grille du joueur sont mémorisées dans le tableau `grille_joueur%()`. Les cases vides contiennent la valeur 0 et les cases occupées par un navire contiennent le numéro du navire. Le schéma suivant est un exemple d'utilisation du tableau:

Le tableau `navires_disponibles%()` contient le nombre de navires de chaque type restant à placer sur la grille. La variable `type_navire%` contient le numéro du type du navire courant. Son contenu est modifié par les options `SOUS-MARINS`, `TORPILLEURS`, `CROISEURS` et `PORTE-AVION`.

La variable `navire_direction%` contient le sens de positionnement des navires (1=horizontal; 2=vertical). Son contenu peut être modifié par les options `NAVIRE HORIZONTAL` et `NAVIRE VERTICAL`

Test de positionnement des navires

Selon leur taille, les navires occupent une zone plus ou moins importante dans la grille. Pour qu'un navire puisse se trouver à un emplacement particulier, il faut que la zone correspondante ne contienne aucun navire, donc que toutes ces cases ne contiennent que la valeur 0. La fonction:

`@tst_grille_vide(l%,c%,,nb_l%,nb_c%)` vérifie que la zone de position (`ligne l%`; `colonne c%`), mesurant `nb_l%` de haut et nb__c% de large est complètement vide. Avant de placer un sous-marin à l'emplacement (5,7), il faut vérifier que la case (5,7) ne contienne pas d'autres navires. Comme aucun navire ne peut se trouver dans les cases adjacentes à un autre navire, il faut également tester les 8 cases entourant la case (5,7). Ces deux opérations peuvent se faire en une seule fois en utilisant la fonction:

`@tst_grille_vide`

sur la zone de position (4,6) et de taille (3,3). Le sous-marin est un cas simple, puisqu'il n'occupe qu'une case. La zone de test a toujours la dimension de 3 cases de haut et de large quel que que soit le sens d'orientation des navires. Ce n'est pas le cas avec les autres types de navires. Il faut alors tenir compte du sens d'orientation de ces derniers pour calculer les dimensions de la zone de test.

La fonction:

`@tst_stockage_navire(l%,c%,n%)`

teste si un navire de type `n%` peut être stocké à partir de la case (l%,c%). Elle vérifie si le navire ne dépasse pas les limites de l'écran et si la zone de stockage est vide. Elle utilise la fonction `@taille_navire` pour connaître la taille du navire à tester, la variable globale `navire_direction%` pour connaître le sens d'orientation des navires et la fonction `@tst_grille_vide` pour tester la zone de stockage.

La procédure:

`@stockage_navire(l%,c%,n%)`

stocke le navire `n%` à partir de la case (`l%,c%`). Elle consulte la variable `navire_direction%` pour connaître le sens d'orientation du navire.

La procédure:

`@visualisation_navire(l%,c%,n%)`

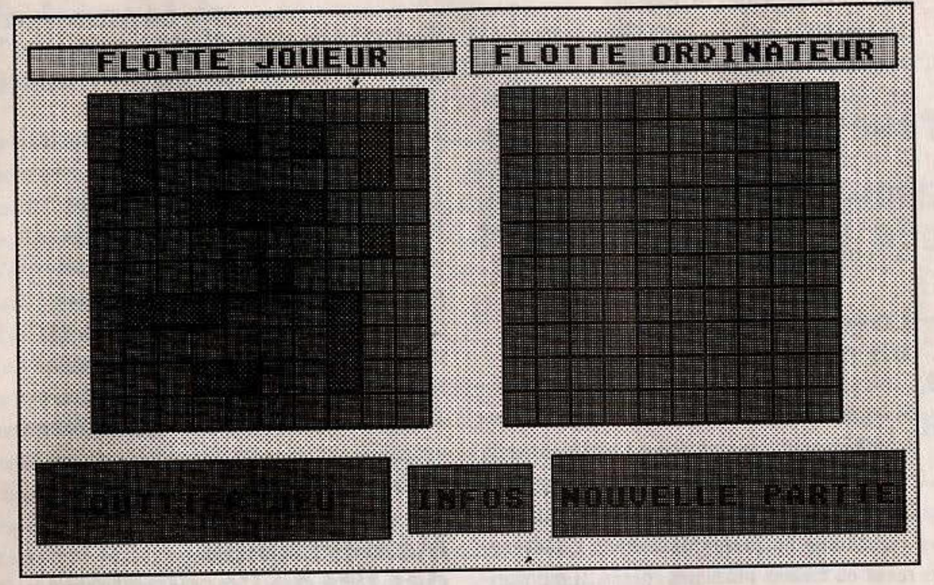
dessine la silhouette du navire `n%` sous la forme d'un rectangle marron dont la taille dépend du type de navire. Les cotés en bas et à droite du rectangle sont encadrés par une ligne noire produisant un effet de relief qui donne l'impression que les navires sont posés sur la grille.

Enlèvement d'un navire

La procédure:

`enleve_navire(l%,c%,navire%)`

efface le navire de type `navire%` se trouvant à la position (`l%,c%`). Le processus d'enlèvement est un peu complexe, car on ne con-



naît pas à priori le sens d'orientation du navire. La routine efface donc systématiquement toutes les cases occupées dans les 4 directions autour de la case (l%,c%). L'effacement ne cesse que lorsque la routine tombe sur une case vide ou sur les bords de la grille. Cette technique possède l'avantage d'effacer un navire en sélectionnant n'importe quelle des cases qu'il occupe. Au fur et à mesure que la routine efface le stockage du navire en mémoire, elle utilise la procédure `@efface_grille` pour effacer le dessin du navire sur la grille.

Gestion de l'interface utilisateur

La routine `@dessin_ecr_pos` dessine l'écran de saisie de la flotte, avec le dessin de la grille et les différentes options. Elle utilise plusieurs sous-routines pour dessiner des éléments particuliers. La gestion de l'interface utilisateur est classique: le programme attend que l'utilisateur clique sur une position `(x,y)`, teste si cette dernière correspond à la grille ou à une option et exécute la routine correspondante. La procédure `@ges_creation` gère la totalité de la phase de création. Elle utilise la procédure `@att_clic` pour attendre que l'utilisateur clique sur la souris.
La procédure @test__creation teste si la position de clic correspond à l'une des options actives, ou à la grille. Elle renvoie un numéro d'identification.
La procédure `@exec_creation` exécute les routines correspondant aux options.

La procédure `@clic_grille_creation` traite le clic sur la grille. Elle détermine la case cliquée, appelle la routine de création de navire si la case est vide ou la routine d'effacement de navire si la case est déjà occupée.

Positionnement de la flotte par l'ordinateur

La routine `@placement_flotte_ordinateur` «fabrique» la flotte de l'ordinateur. Les navires sont stockés dans le tableau `grille_ordinateur%()`. Le placement des navires se fait de la manière suivante: le programme détermine aléatoirement une position et une orientation. Si cette position ne convient pas, il calcule une nouvelle position. La fonction `@tst_navire_ordinateur(l%,c%,n%,d%)` vérifie si un navire n% et d'orientation d% peut être présent à la position `(l%,c%)`. La procédure `@stockage_navire_ordinateur(l%,c%,n%,d%)` écrit le n% à la position `(l%,c%)`, en tenant compte de son orientation d%.

Contrairement aux navires du joueur, les navires de l'ordinateur possèdent chacun un numéro de référence (de 1 à 10). Le porte-avions a le numéro 1. Les croiseurs ont les numéros 2 et 3. Les torpilleurs les numéros 4,5 et 6. Les sous-marins portent les numéros 7,8,9 et 10. Cette mémorisation est indispensable car le joueur n'agit pas toujours de manière logique. Il peut, par exemple, endommager plusieurs navires sans les couler, afin de se réserver le plaisir de les détruire en fin de jeu. C'est un cas rare, mais qui peut se produire. Le programme doit donc stocker des informations précises sur l'état de chaque navire et pour cela, donne un numéro d'identification à chaque navire. Les navires du joueur n'ont pas de numéro d'identification, mais uniquement un type de navire. Cela vient du fait que lorsque l'ordinateur repère un navire, il s'acharne sur lui tant qu'il n'est pas coulé. Le programme se contente donc de ne stocker en mémoire que les données du navire courant.
Il est plus facile de placer un sous-marin dans une grille encombré de gros navires, que de placer un porte-avions dans une grille occupé par des sous-marins et des navires de petites tailles. C'est pourquoi la routine de «fabrication» de la flotte place en premier le porte-avions, puis les croiseurs, les torpilleurs et enfin les sous-marins.

Phase de combat

Le combat est une succession de tirs. Le joueur tire sur la flotte de l'ordinateur qui riposte en tirant sur les navires du joueur. L'écran de jeu contient le dessin de la flotte du joueur, une grille vide pour la flotte de l'ordinateur (les navires ne sont évidemment pas affichés), une option `NOUVELLE PARTIE`, une option `INFOS` et une option `QUITTER PROGRAMME` La procédure `@gestion_jeu`

TABLEAU 2

0	0	0	0	0
0	0	0	0	0
0	0	X	0	0
0	0	0	0	0
0	0	0	0	0

<= X = Position du Sous-marin

gère l'alternance des combats. L'affrontement ne cesse que si une des variables fin_prg%, joueur_gagnant%, ordinateur_gagnant% ou nouvelle_partie% passe à 1. Ces variables sont les conditions de sortie de la routine (voir tableau 2).

Gestion des tirs du joueur

Pour effectuer un tir, le joueur doit cliquer sur une case de la grille de l'ordinateur. Si cette case ne contient aucun navire, il s'y affiche une petite croix noire. Dans le cas contraire, le programme dessine un petit cercle rouge pour symboliser le fait que le navire soit touché. Si le navire est complètement détruit, toutes ses cases sont recouvertes par des croix rouges.

La routine @gestion_joueur s'occupe de gérer les actions du joueur. Celles-ci peuvent être le tir sur la flotte ennemi, mais aussi l'appel d'une des fonctions disponibles sur l'écran (INFOS, QUITTER PRG et NOUVELLE PARTIE). La routine est essentiellement une boucle qui s'exécute tant que le joueur ne tire pas sur la flotte ennemie, c'est-à-dire tant que la variable tir_joueur% contient la valeur 0. Cette variable est un indicateur d'état. Le programme passe la plupart de son temps à attendre un clic souris. Si la position de clic correspond à une des options de l'écran de combat, il exécute la routine correspondante.

La fonction @identif_zone(xm%,ym%) renvoie le numéro de la zone écran correspondant à la position (xm%,ym%). La valeur 1 correspond à un clic sur la grille de tir du joueur.

La procédure @tir_joueur(xm%,ym%) s'exécute lorsque le joueur clique sur la grille de tir du joueur. Les variables xm% et ym% correspondent à la position de clic. La routine commence par déterminer la ligne et la colonne de la case cliquée grâce à la routine @identif_case_tir, puis vérifie si cette case est occupée par un navire. Cette opération se fait en lisant le contenu du tableau grille_ordinateur%(). Si la case contient la valeur 0, il n'y a pas de navire à cet endroit. Si c'est un nombre compris entre 1 et 10, c'est le numéro d'un bateau. Si c'est un nombre négatif, cela signifie que l'ordinateur a déjà tiré sur cette case.

Le joueur tire dans l'eau

Si le joueur ne touche aucun navire, le programme affiche une petite croix noire à l'emplacement de la case. Le travail graphique est effectué par la routine:
@aff_crois_vide(l%,c%).

Le logiciel mémorise le fait que le joueur a tiré sur cette case en écrivant la valeur négative -10 dans le tableau grille_ordinateur%(). La variable tir_joueur% est mise à 1 afin de signaler à la routine @gestion_joueur que le joueur a effectué son tir et qu'il est temps que le programme tire sur la flotte du joueur.

Le joueur touche un navire

Si le joueur touche un navire, la variable navire% contient le numéro du bâtiment et la variable type_navire% contient son type (sous_marin, torpilleur, croiseur ou porte_avion). Le programme traite cet événement en faisant plusieurs choses. Il mémorise le tir en écrivant un nombre négatif dans le tableau grille_ordinateur%(), utilise la routine @aff_impact pour afficher un petit cercle rouge à l'emplacement de la case visée et incrémente le compteur d'impacts du navire. Celui-ci est stocké dans le tableau nb_impacts%() qui contient le nombre de tirs encaissés par chaque navire de la flotte de l'ordinateur. La position de la case est ensuite mémorisée avec la procédure @memorisation_impact(l%,c%).

Cette dernière stocke la ligne et la colonne de la case dans le tableau pos_impact%(). Ces données serviront ultérieurement lorsque le logiciel dessinera la forme du navire coulé.

Une fois toutes ces opérations réalisées, le jeu de bataille navale vérifie si le tir qui vient d'être traité ne vient pas de couler le navire. Pour ce faire, il compare la longueur du bâtiment avec le nombre de coups reçus. Si ces nombres sont les mêmes, le navire est coulé. La variable pertes_ordinateur% est alors incrémentée d'une unité. Si 10 bâtiments ont été coulés, cela signifie que le joueur a anéanti la totalité de la flotte de l'ordinateur. La variable joueur_gagnant% est mise à 1. Les navires coulés doivent être représentés graphiquement. C'est la tâche de la fonction @aff_navire_coule(navire%) qui affiche des grandes croix rouges sur les cases du navire navire%. Ces cases ont été préalablement mémorisées dans le tableau pos_impact%() par la routine @memorisation_impact(l%,c%). Une fois le traitement du tir effectué, la variable tir_joueur% est mise à 1 afin de signaler que le joueur vient de jouer et que le programme peut jouer à son tour.

Gestion des tirs de l'ordinateur

La routine @gestion_ordinateur gère le tir de l'ordinateur contre la flotte du joueur. Deux états sont possibles: aucun navire n'est repéré et le tir se fait au hasard, ou un navire est repéré et le tir se fait méthodiquement pour achever le bâtiment. L'état courant du tir est mémorisé dans la variable navire_en_vue%. Si un navire est repéré, cette variable contient 1. Dans le cas contraire, elle contient 0. Cette routine est essentiellement un aiguillage entre les procédures gérant les cas de tirs possibles.

Mémorisation des tirs de l'ordinateur

Comme nous l'avons vu plus haut, les navires du joueur sont stockés dans le tableau grille_joueur%() et les navires de l'ordinateur dans le tableau grille_ordinateur%(). Afin de ne pas tirer sur une case déjà visée, l'ordinateur doit garder une trace de ses tirs. C'est la tâche du tableau tir_ordinateur%(). Lorsque le programme tire sur une case, il écrit la valeur 1 dans tir_ordinateur%().

Tir aléatoire

La routine @tir_aleatoire gère le tir de l'ordinateur lorsque aucun navire n'est repéré. Les tirs se font alors au hasard, dans l'espoir d'endommager ou de couler les bâtiments du joueur. Cette routine utilise une boucle DO-LOOP afin de déterminer une case où aucun tir n'a été effectué. La recherche se fait en calculant aléatoirement des positions et en utilisant le tableau tir_ordinateur%() pour vérifier si un tir a déjà été effectué sur cette case. Une fois la case choisie, le programme lit son contenu dans le tableau grille_joueur%() -contenant la flotte du joueur- et teste sa valeur. Le traitement suivant dépend du chiffre lu.

Tir aléatoire raté

Si la case de tir contient 0, il n'y a aucun navire à cet endroit. Le programme visualise l'échec en affichant un petit cercle bleu à l'emplacement de la case. Il modifie ensuite

le tableau `tir_ordinateur%()` pour ne plus tirer à cet endroit.

TABLEAU 3

0	0	0	0	0
0	.	.	.	0
0	.	X	.	0
0	.	.	.	0
0	0	0	0	0

X = Position du Sous-marin

. = Cases ou il ne peut y avoir de navires

Traitement de l'impact sur un sous-marin

Le cas où la case de tir contient le numéro d'un sous-marin est simple à gérer puisque ce type de navire n'occupe qu'une case et coule au moment même où il est touché.

Le programme incrémente la variable `pertes_joueur%` et utilise la procédure `@impact_joueur` pour représenter visuellement la perte du navire en affichant une grande croix rouge à l'emplacement de la case de tir. Nous avons vu plus haut qu'il était impossible qu'un navire en touche un autre. Il ne peut donc avoir de navires dans les 8 cases adjacentes au sous-marin. En marquant ces 8 cases, on évite que l'ordinateur ne tire dessus inutilement.

La routine `@marquage_sous_marin(l%,c%)` marque les 8 cases entourant la case `(l%,c%)` comme étant déjà testées. Elle est écrite de manière à marquer même les cases qui se trouvent au bord de la grille (voir tableau 3).

Traitement de l'impact sur un navire

Lorsque l'ordinateur touche un navire composé d'au moins deux cases (torpilleur, croiseur ou porte-avions), il effectue les traitements graphiques habituels, stocke la position de tir dans les variables `l_cible%` et `c_cible%` et mémorise le fait qu'un navire ennemi soit touché en écrivant le type de navire dans la variable `navire_en_vue%`. La valeur de cette variable détermine la stratégie de tir du programme. La valeur `0` signifie qu'aucun navire du joueur n'est touché et qu'il faut tirer au hasard sur les cases non explorées. Tandis qu'une valeur différente de `0` implique qu'un navire ennemi est touché et qu'il faut donc lui tirer dessus. La discrimination entre ces deux stratégies s'effectue à l'aide de la procédure `@gestion_ordinateur`.

Tir sur un navire repéré

La routine `@tir_sur_navire` s'occupe de tirer sur les navires du joueur ayant été repérés à la suite d'un tir au hasard. C'est l'une des tâches les plus complexes du jeu, car il faut d'abord déterminer la direction du navire (horizontal ou vertical) pour le détruire totalement. Le programme détermine aléatoirement une direction de tir avec la procédure `@new_direction_tir`, direction mémorisée dans la variable `cible_direction%`. Les procédures `@tir_gauche`, `@tir_droit`, `@tir_haut` et `@tir_bas` gèrent chacune de ses directions de tirs. Lorsque le navire est coulé, c'est-à-dire lorsque les dommages qu'il a encaissés ont la même valeur que son nombre de cases, la valeur `0` est écrite dans la variable `navire_en_vue%`. Le logiciel sait alors qu'il va devoir recommencer à tirer au hasard, dans l'espoir de toucher d'autre navire.

Amélioration du programme

Le programme de bataille navale tel qu'il est actuellement est une version minimale. Vous pouvez considérablement l'améliorer. Des effets sonores peuvent signaler au joueur qu'un navire est touché ou qu'il coule. Au lieu de prendre des carrés de couleur pour représenter les navires, vous pouvez utiliser des silhouettes de bateau.

Le listing du programme est disponible en téléchargement, mais aussi sur la disquette fournie avec ce numéro d'Atari Magazine.

Patrick Leclercq ■

```
' ******************************
' *  BATAILLE NAVALE           *
' ******************************
' *  (C) 1992 PATRICK LECLERCQ *
' *  (C) 1992 ARTIPRESSE       *
' ******************************
' *  Ecrit en GFA BASIC 3.xx   *
' ******************************
RESERVE 100000
OPTION BASE 1
DIM grille_joueur%(10,10)
DIM grille_ordinateur%(10,10)
DIM tir_ordinateur%(10,10)
'
' ****************************************
' *  CONSTANTES POUR LES TYPES DE NAVIRES *
' ****************************************
sous_marin%=1
torpilleur%=2
croiseur%=3
porte_avion%=4
'
' ******************************
' *  NB DE NAVIRES DISPONIBLES *
' ******************************
DIM navires_disponibles%(4)
'
' ********************************************
' *  TABLEAU CONTENANT LE NB DE COUPS SUBIS *
' *  PAR LES NAVIRES DE L'ORDINATEUR         *
' ********************************************
DIM nb_impacts%(10)
'
' ************************************************
' *  TABLEAU CONTENANT LES POSITIONS DES NAVIRES  *
' *  DE L'ORDINATEUR TOUCHE PAR LE JOUEUR         *
' *  pos_impact%(n,i,p)                           *
' *  n = numéro de navire (1 à 10)                *
' *  i = numéro impact (1 à 4)                    *
' *  p = ligne ou colonne (1=ligne; 2=colonne)    *
' ************************************************
DIM pos_impact%(10,4,2)
'
' ****************************************
' *  MEMORISATION DES DIRECTIONS DE TIR *
' *  DEJA UTILISEES PAR LE PROGRAMME    *
' ****************************************
DIM test_direction_tir%(4)
'
' ****************************************
' *  MEMORISATION DES POSITIONS DU NAVIRE *
' *  TOUCHE PAR LE PROGRAMME               *
' ****************************************
DIM pos_navire_en_vue%(4,2)
@main
```

```
END
'
' ********************************
' *                              *
' *   FONCTIONS GRAPHIQUES DE BASE *
' *                              *
' ********************************
' ********************************
' *  AFFICHAGE D'UN RECTANGLE PLEIN *
' ********************************
PROCEDURE rect(px%,py%,tx%,ty%,c%)
  LOCAL px2%,py2%
  '
  px2%=px%+tx%-1           ! CALCUL POS X EXTREME RECT
  py2%=py%+ty%-1           ! CALCUL POS Y EXTREME RECT
  DEFFILL c%               ! DEFINITION COULEUR DU FOND
  PBOX px%,py%,px2%,py2%   ! TRACE COULEUR DE FOND
RETURN
'
' ****************************
' *  AFFICHAGE D'UN CADRE VIDE *
' ****************************
PROCEDURE cadre(px%,py%,tx%,ty%,c%)
  LOCAL px2%,py2%
  '
  px2%=px%+tx%-1           ! CALCUL POS X EXTREME RECT
  py2%=py%+ty%-1           ! CALCUL POS Y EXTREME RECT
  COLOR c%                 ! DEFINITION COULEUR DU FOND
  BOX px%,py%,px2%,py2%    ! TRACE COULEUR DE FOND
RETURN
'
' ****************************
' *  AFFICHAGE D'UNE BOITE VIDE *
' ****************************
PROCEDURE boite(px%,py%,tx%,ty%,fond%,contour%)
  LOCAL px2%,py2%
  '
  px2%=px%+tx%-1           ! CALCUL POS X EXTREME CADRE
  py2%=py%+ty%-1           ! CALCUL POS Y EXTREME CADRE
  DEFFILL fond%            ! DEFINITION COULEUR DU FOND
  PBOX px%,py%,px2%,py2%   ! TRACE CADRE DE FOND
  COLOR contour%           ! DEFINITION COULEUR CONTOUR
  BOX px%,py%,px2%,py2%    ! TRACE DU CONTOUR
RETURN
'
' ************************************************
' *  AFFICHAGE D'UN CADRE DE SELECTION AVEC MESSAGE *
' ************************************************
PROCEDURE option(px%,py%,tx%,ty%,f%,c%,m$)
  LOCAL xm%,ym%
  '
  xm%=px%+(tx%-(LEN(m$)*8))/2   '! CALCUL POSITION X
MESSAGE
  ym%=py%+7+(ty%-8)/2
  DEFTEXT noir
```

```
  @boite(px%,py%,tx%,ty%,f%,c%) ! TRACE DU CADRE
  TEXT xm%,ym%,m$               ! AFFICHAGE DU MESSAGE
RETURN
'
' *******************************
' *  ATTENTE PAS DE CLIC SOURIS  *
' *******************************
> PROCEDURE att0clic
  DO
    EXIT IF MOUSEK=0
  LOOP
RETURN
'
' *****************************
' *  ATTENTE D'UN CLIC SOURIS *
' *****************************
> PROCEDURE wait_clic
  @att0clic
  DO
    EXIT IF MOUSEK<>0
  LOOP
  @att0clic
RETURN
'
' *****************************
' *  ATTENTE CLIC SOURIS      *
' *  AVEC LECTURE ETAT SOURIS *
' *****************************
PROCEDURE att_clic(VAR xm%,ym%,km%)
  DO
    MOUSE xm%,ym%,km%
    EXIT IF km%<>0
  LOOP
RETURN
'
' ****************************************
' *  TEST SI CLIC SUR UNE ZONE GRAPHIQUE  *
' ****************************************
> FUNCTION tstzone(xm%,ym%,px%,py%,tx%,ty%)
LOCAL px2%,py2%,rep%
'
rep%=0
px2%=px%+tx%-1
py2%=py%+ty%-1
IF (xm%>=px%) AND (xm%<=px2%)
  IF (ym%>=py%) AND (ym%<=py2%)
    rep%=1
  ENDIF
ENDIF
RETURN rep%
ENDFUNC
'
' ******************************************
' *                                       *
' *  DESSIN DES ELEMENTS GRAPHIQUES DU JEU  *
' *                                       *
' ******************************************
' ********************************
' *  DESSIN D'UNE GRILLE DE JEU  *
' ********************************
> PROCEDURE dessin_grille(px%,py%)
LOCAL ligne%,colonne%
LOCAL xligne%,yligne%
LOCAL xligne2%,yligne2%
'
' ************************
' *  REMPLISSAGE DU FOND  *
' ************************
@rect(px%,py%,tx_case%*10,ty_case%*10,bleu)
'
' **********************
' *  LIGNES VERTICALES  *
' **********************
xligne%=px%
yligne%=py%
xligne2%=xligne%+(tx_case%*10)-1
FOR ligne%=1 TO 11
  COLOR noir
  LINE xligne%,yligne%,xligne2%,yligne%
  ADD yligne%,ty_case%
NEXT ligne%
'
' ************************
' *  LIGNES HORIZONTALES  *
' ************************
xligne%=px%
yligne%=py%
yligne2%=yligne%+(ty_case%*10)-1
FOR colonne%=1 TO 11
  COLOR noir
  LINE xligne%,yligne%,xligne%,yligne2%
  ADD xligne%,tx_case%
NEXT colonne%
RETURN
'
' ****************************
' *  EFFACEMENT D'UNE CASE     *
' *  DE LA GRILLE DE CREATION  *
' ****************************
> PROCEDURE efface_case(l%,c%)
LOCAL px%,py%
'
px%=(c%-1)*tx_case%+8
py%=(l%-1)*ty_case%+51
@boite(px%,py%,tx_case%+1,ty_case%+1,bleu,noir)
RETURN
'
' ****************************************
' *  AFFICHAGE DU TYPE DE NAVIRE COURANT  *
' ****************************************
```

```
> PROCEDURE affiche_type_navire(navire%)
HIDEM
IF type_navire%<>0
  SELECT type_navire%
  CASE sous_marin%
    @aff_option(sous_marin%,gris)
  CASE torpilleur%
    @aff_option(torpilleur%,gris)
  CASE croiseur%
    @aff_option(croiseur%,gris)
  CASE porte_avion%
    @aff_option(porte_avion%,gris)
  ENDSELECT
ENDIF
SELECT navire%
CASE sous_marin%
  @aff_option(sous_marin%,rouge)
CASE torpilleur%
  @aff_option(torpilleur%,rouge)
CASE croiseur%
  @aff_option(croiseur%,rouge)
CASE porte_avion%
  @aff_option(porte_avion%,rouge)
ENDSELECT
SHOWM
RETURN
'
' *******************************************
' *  AFFICHAGE DU NB DE NAVIRES DISPONIBLES *
' *******************************************
> PROCEDURE aff_navires_disponibles(n%)
LOCAL px%,py%,m$
'
m$=STR$(navires_disponibles%(n%))
SELECT n%
CASE sous_marin%
  px%=288
  py%=52
CASE torpilleur%
  px%=288
  py%=76
CASE croiseur%
  px%=288
  py%=99
CASE porte_avion%
  px%=288
  py%=123
ENDSELECT
@option(px%,py%,24,13,gris,noir,m$)
RETURN
'
' *******************************
' *  AFFICHAGE DIRECTION NAVIRE  *
' *******************************
> PROCEDURE aff_direction_navire
IF navire_direction%=1
  m$="NAVIRE HORIZONTAL"
  @option(136,147,176,13,violet,noir,m$)
  m$="NAVIRE VERTICAL"
  @option(136,159,176,13,jaune,noir,m$)
ELSE
  m$="NAVIRE HORIZONTAL"
  @option(136,147,176,13,jaune,noir,m$)
  m$="NAVIRE VERTICAL"
  @option(136,159,176,13,violet,noir,m$)
ENDIF
RETURN
'
' ************************************
' *  AFFICHAGE OPTION "COMMENCER JEU"  *
' ************************************
> PROCEDURE aff_flotte_ok
LOCAL m$
'
@boite(136,147,176,25,violet,noir)
m$="COMMENCER JEU"
@option(137,153,174,13,violet,violet,m$)
RETURN
'
' ****************************************
' *  AFFICHAGE DES SELECTEURS D'OPTIONS *
' ****************************************
' *  n%: type d'option                   *
' *  c%: couleur de fond du texte        *
' ****************************************
> PROCEDURE aff_option(n%,c%)
SELECT n%
CASE sous_marin%
  @option(136,52,144,13,c%,noir,"SOUS-MARINS")
CASE torpilleur%
  @option(136,76,144,13,c%,noir,"TORPILLEURS")
CASE croiseur%
  @option(136,99,144,13,c%,noir,"CROISEURS")
CASE porte_avion%
  @option(136,123,144,13,c%,noir,"PORTE-AVIONS")
ENDSELECT
RETURN
'
' ***************************
' *  ECRAN DE POSITIONNEMENT  *
' *  DE LA FLOTTE DU JOUEUR   *
' ***************************
> PROCEDURE dessin_ecr_pos
LOCAL m$
'
DEFLINE ,1
@rect(0,0,320,200,blanc)
@option(8,28,120,13,vert,noir,"FLOTTE JOUEUR")
@dessin_grille(8,51)
m$="NAVIRES DISPONIBLES"
```

```
@option(136,28,176,13,vert,noir,m$)
@aff_option(sous_marin%,rouge)
@aff_navires_disponibles(sous_marin%)
@aff_option(torpilleur%,gris)
@aff_navires_disponibles(torpilleur%)
@aff_option(croiseur%,gris)
@aff_navires_disponibles(croiseur%)
@aff_option(porte_avion%,gris)
@aff_navires_disponibles(porte_avion%)
@aff_direction_navire
RETURN
'
' ***********************************************
' *  TEST SI UNE ZONE DE LA GRILLE DE CREATION  *
' *  EST VIDE.                                  *
' ***********************************************
' *  l%,c%: position de la zone à tester        *
' *  nb_l%,nb_c%: dimension de la zone          *
' ***********************************************
> FUNCTION tst_grille_vide(l%,c%,nb_l%,nb_c%)
LOCAL l0%,c0%,rep%
'
rep%=TRUE
FOR l0%=l% TO l%+nb_l%-1
FOR c0%=c% TO c%+nb_c%-1
  IF (c0%>0) AND (c0%<11) AND (l0%>0) AND (l0%<11)
    IF grille_joueur%(l0%,c0%)<>0
      rep%=FALSE
    ENDIF
  ENDIF
NEXT c0%
NEXT l0%
'
RETURN rep%
ENDFUNC
'
' *************************************
' *  IDENTIFICATION DE LA CASE        *
' *  SELECTIONNEE PAR LE JOUEUR       *
' *  VALABLE UNIQUEMENT POUR LA       *
' *  GRILLE DE CREATION DE LA FLOTTE  *
' *************************************
> PROCEDURE identif_case_creation(xm%,ym%,VAR l%,c%)
LOCAL px%,py%
'
px%=xm%-8
py%=ym%-51
l%=(py%/ty_case%)+1
c%=(px%/tx_case%)+1
RETURN
'
' *****************************************
' *  LECTURE DU TYPE DE NAVIRE SITUE DANS  *
' *  UNE CASE DE LA GRILLE DE CREATION     *
' *****************************************
FUNCTION lec_case(l%,c%)
RETURN grille_joueur%(l%,c%)
ENDFUNC
'
' *******************************************
' *  DETERMINATION DE LA TAILLE D'UN NAVIRE  *
' *******************************************
> FUNCTION taille_navire(n%)
LOCAL taille%
'
SELECT n%
CASE sous_marin%
taille%=1
CASE torpilleur%
taille%=2
CASE croiseur%
taille%=3
CASE porte_avion%
taille%=4
ENDSELECT
RETURN taille%
ENDFUNC
'
' **********************************
' *  TEST SI UN NAVIRE PEUT ETRE    *
' *  STOCKE DANS UNE CASE PRECISE   *
' **********************************
> FUNCTION tst_stockage_navire(l%,c%,n%)
LOCAL rep%,taille%
'
rep%=TRUE
taille%=@taille_navire(n%)
' ***********************
' *  NAVIRE HORIZONTAL  *
' ***********************
IF navire_direction%=1
IF c%>(11-taille%) ! TEST SI DEPASSEMENT GRILLE
rep%=FALSE
ENDIF
' *  TEST SI NAVIRE DANS CASES ADJACENTES  *
IF NOT (@tst_grille_vide(l%-1,c%-1,3,taille%+2))
rep%=FALSE
ENDIF
ELSE
' *********************
' *  NAVIRE VERTICAL  *
' *********************
IF l%>(11-taille%) ! TEST SI DEPASSEMENT GRILLE
rep%=FALSE
ENDIF
' *  TEST SI NAVIRE DANS CASES ADJACENTES  *
IF NOT (@tst_grille_vide(l%-1,c%-1,taille%+2,3))
rep%=FALSE
ENDIF
ENDIF
```

```
'
RETURN rep%
ENDFUNC
'
' ***********************************
' *  STOCKAGE D'UN NAVIRE EN MEMOIRE  *
' ***********************************
' *  l%,c%: position du navire        *
' *  n%:    type de navire            *
' ***********************************
> PROCEDURE stockage_navire(l%,c%,n%)
LOCAL l0%,c0%,taille%
'
taille%=@taille_navire(n%)
IF navire_direction%=1  ! NAVIRE HORIZONTAL
FOR c0%=c% TO c%+taille%-1
grille_joueur%(l%,c0%)=n%
NEXT c0%
ELSE                     ! NAVIRE VERTICAL
FOR l0%=l% TO l%+taille%-1
grille_joueur%(l0%,c%)=n%
NEXT l0%
ENDIF
RETURN
'
' *******************************************
' *  VISUALISATION DU STOCKAGE D'UN NAVIRE  *
' *******************************************
> PROCEDURE visualisation_navire(l%,c%,n%)
LOCAL px%,py%,taille%
'
taille%=@taille_navire(n%)
px%=(c%-1)*tx_case%+8
py%=(l%-1)*ty_case%+51
IF navire_direction%=1
' *  NAVIRE HORIZONTAL  *
@boite(px%,py%,taille%*tx_case%,ty_case%,marron,noir)
ELSE
' *  NAVIRE VERTICAL  *
@boite(px%,py%,tx_case%,taille%*ty_case%,marron,noir)
ENDIF
RETURN
'
' *******************************************
' *  TEST SI LA FLOTTE DU JOUEUR EST PRETE  *
' *******************************************
> FUNCTION tst_flotte
LOCAL n%
'
n%=navires_disponibles%(sous_marin%)
ADD n%,navires_disponibles%(torpilleur%)
ADD n%,navires_disponibles%(croiseur%)
ADD n%,navires_disponibles%(porte_avion%)
IF n%=0
RETURN TRUE
ELSE
RETURN FALSE
ENDIF
ENDFUNC
'
' ****************************************
' *  SELECTION ET CREATION D'UN NAVIRE  *
' ****************************************
> PROCEDURE creation_navire(l%,c%,n%)
IF navires_disponibles%(n%)<>0
IF @tst_stockage_navire(l%,c%,n%)
HIDEM
@visualisation_navire(l%,c%,n%)
@stockage_navire(l%,c%,n%)
DEC navires_disponibles%(n%)
@aff_navires_disponibles(n%)
' *  TEST SI LA FLOTTE EST COMPLETE
IF @tst_flotte
@aff_flotte_ok
ENDIF
SHOWM
ENDIF
ENDIF
RETURN
'
' ******************************
' *  EFFACEMENT D'UN NAVIRE     *
' *  SUR LA GRILLE DE CREATION  *
' ******************************
> PROCEDURE enleve_navire(l%,c%,navire%)
LOCAL l0%,c0%
'
INC navires_disponibles%(navire%)
HIDEM
IF navire%<>type_navire%
@affiche_type_navire(navire%)
ENDIF
@aff_navires_disponibles(navire%)
type_navire%=navire%
' *  EFFACEMENT DU NAVIRE  *
grille_joueur%(l%,c%)=0
@efface_case(l%,c%)
' *  EFFACEMENT VERS LA GAUCHE  *
c0%=c%
DO
DEC c0%
EXIT IF c0%=0
EXIT IF grille_joueur%(l%,c0%)=0
grille_joueur%(l%,c0%)=0
@efface_case(l%,c0%)
LOOP
' *  EFFACEMENT VERS LA DROITE  *
c0%=c%
DO
INC c0%
```

```
EXIT IF c0%=11
EXIT IF grille_joueur%(l%,c0%)=0
grille_joueur%(l%,c0%)=0
@efface_case(l%,c0%)
LOOP
' *  EFFACEMENT VERS LE HAUT    *
l0%=l%
DO
DEC l0%
EXIT IF l0%=0
EXIT IF grille_joueur%(l0%,c%)=0
grille_joueur%(l0%,c%)=0
@efface_case(l0%,c%)
LOOP
' *  EFFACEMENT VERS LE BAS      *
l0%=l%
DO
INC l0%
EXIT IF l0%=11
EXIT IF grille_joueur%(l0%,c%)=0
grille_joueur%(l0%,c%)=0
@efface_case(l0%,c%)
LOOP
' *  TEST SI L'OPTION "COMMENCER JEU" EXISTE
' *  IL FAUT L'ENLEVER DE L'ECRAN
IF @tst_flotte=FALSE
@aff_direction_navire
ENDIF
SHOWM
RETURN
'
' ***************************
' *  TRAITEMENT CLIC SUR    *
' *  LA GRILLE DE CREATION  *
' ***************************
> PROCEDURE clic_grille_creation(xm%,ym%)
LOCAL l%,c%,navire%
'
@identif_case_creation(xm%,ym%,l%,c%)
navire%=@lec_case(l%,c%)
IF navire%=0  ! PAS DE NAVIRE
@creation_navire(l%,c%,type_navire%)
ELSE          ! PRESENCE D'UN NAVIRE A ENLEVER
@enleve_navire(l%,c%,navire%)
ENDIF
RETURN
'
' ********************************
' *  CLIC SUR NAVIRE HORIZONTAL  *
' ********************************
> PROCEDURE clic_navire_horizontal
IF navire_direction%=2
navire_direction%=1
HIDEM
@aff_direction_navire
SHOWM
ENDIF
RETURN
'
' ******************************
' *  CLIC SUR NAVIRE VERTICAL  *
' ******************************
> PROCEDURE clic_navire_vertical
IF navire_direction%=1
navire_direction%=2
HIDEM
@aff_direction_navire
SHOWM
ENDIF
RETURN
'
' *************************************
' *  CLIC SUR UNE OPTION DE SELECTION  *
' *  D'UN TYPE DE NAVIRE               *
' *************************************
> PROCEDURE clic_select_navire(n%)
IF type_navire%<>n%
@affiche_type_navire(n%)
type_navire%=n%
ENDIF
RETURN
'
' ****************************************
' *  EXECUTION DES COMMANDES DE CREATION  *
' ****************************************
> PROCEDURE exec_creation(n%)
SELECT n%
CASE 3
@clic_select_navire(sous_marin%)
CASE 4
@clic_select_navire(torpilleur%)
CASE 5
@clic_select_navire(croiseur%)
CASE 6
@clic_select_navire(porte_avion%)
CASE 7
@clic_navire_horizontal
CASE 8
@clic_navire_vertical
ENDSELECT
RETURN
'
' ******************************************
' *  TEST SI CLIC SUR UNE FONCTION DU MENU  *
' *  DE CREATION DE LA FLOTTE DU JOUEUR     *
' ******************************************
' *  0 = Pas de clic                        *
' *  1 = Tous les navires sont placés       *
' *  2 = Clic sur grille de jeu             *
' *  3 = Clic sur option sous-marin         *
```

```
' *  4 = Clic sur option Torpilleurs          *
' *  5 = Clic sur option Croiseurs            *
' *  6 = Clic sur option Porte-avions         *
' *  7 = Clic sur Pos horizontale             *
' *  8 = Clic sur Pos Verticale               *
' *  9 = Clic sur option "COMMENCER JEU"      *
' ******************************************
> FUNCTION test_creation(xm%,ym%)
LOCAL rep%
'
rep%=0
IF @tstzone(xm%,ym%,8,51,120,120)
rep%=2
ENDIF
IF @tstzone(xm%,ym%,136,52,176,13)
rep%=3
ENDIF
IF @tstzone(xm%,ym%,136,76,176,13)
rep%=4
ENDIF
IF @tstzone(xm%,ym%,136,99,176,13)
rep%=5
ENDIF
IF @tstzone(xm%,ym%,136,123,176,13)
rep%=6
ENDIF
' *  TEST SI LES OPTIONS POSITION NAVIRE
' *  SONT ACTIVES
IF @tst_flotte=FALSE
IF @tstzone(xm%,ym%,136,147,176,13)
rep%=7
ENDIF
IF @tstzone(xm%,ym%,136,159,176,13)
rep%=8
ENDIF
ELSE
' * TEST SUR ZONE "COMMENCER JEU" SI ACTIVE
IF @tstzone(xm%,ym%,136,147,176,25)
rep%=9
ENDIF
ENDIF
RETURN rep%
ENDFUNC
'
' ****************************
' *  GESTION DE LA CREATION  *
' *  DE LA FLOTTE DU JOUEUR  *
' ****************************
> PROCEDURE ges_creation
LOCAL xm%,ym%,km%
'
DO
@att_clic(xm%,ym%,km%)
'
clic%=@test_creation(xm%,ym%)
'
IF clic%<>0  ! TEST SI CLIC SUR UNE OPTION
@exec_creation(clic%)
@att0clic
ENDIF
'
IF clic%=2  ! TEST SI CLIC SUR GRILLE DE CREATION
@clic_grille_creation(xm%,ym%)
@att0clic
ENDIF
'
EXIT IF clic%=9  ! TEST SI CLIC "COMMENCER JEU"
LOOP
' MEMORISATION IMAGE FLOTTE JOUEUR
GET 8,51,128,171,flotte_joueur$
RETURN
'
'
'
' *********************************
' *                               *
' *  ROUTINES DE GESTION DU JEU  *
' *                               *
' *********************************
'
' **********************************
' *  AFFICHAGE DE L'ECRAN DE JEU  *
' **********************************
PROCEDURE aff_jeu
LOCAL m$
'
@rect(0,0,320,200,blanc)
m$="FLOTTE JOUEUR"
@option(5,11,152,13,vert,noir,m$)
m$="FLOTTE ORDINATEUR"
@option(162,11,152,13,vert,noir,m$)
PUT 26,28,flotte_joueur$  ! IMAGE GRILLE JOUEUR
@dessin_grille(172,28)
m$="QUITTER JEU"
@option(6,159,126,30,gris,noir,m$)
m$="INFOS"
@option(138,162,45,25,gris,noir,m$)
m$="NOUVELLE PARTIE"
@option(189,159,126,30,gris,noir,m$)
RETURN
'
' ************************************************
' *  DESSIN D'UN IMPACT SUR UN NAVIRE DU JOUEUR  *
' ************************************************
> PROCEDURE impact_joueur(l%,c%)
LOCAL xc%,yc%
'
xc%=(c%-1)*tx_case%+26
yc%=(l%-1)*ty_case%+28
COLOR rouge
```

```
DEFFILL rouge
PCIRCLE xc%+6,yc%+6,4
DEFLINE ,3
LINE xc%,yc%,xc%+11,yc%+11
LINE xc%,yc%+11,xc%+11,yc%
RETURN
'
' **********************************
' *  L'ORDINATEUR TIRE DANS L'EAU  *
' **********************************
> PROCEDURE aff_tir_eau(l%,c%)
LOCAL xc%,yc%
'
xc%=(c%-1)*tx_case%+26
yc%=(l%-1)*ty_case%+28
'
DEFFILL bleu3
PCIRCLE xc%+6,yc%+6,4
RETURN
'
' ____________________________________________
' ********************************************
' *  MEMORISATION D'UNE DES CASES OCCUPEES  *
' *  PAR LE NAVIRE SUBIS AU TIR DU PRG      *
' ********************************************
> PROCEDURE memorise_case_ennemi(l%,c%)
pos_navire_en_vue%(dommages_navire%,1)=l%
pos_navire_en_vue%(dommages_navire%,2)=c%
RETURN
'
' _____________________________________
' *************************************
' *  LE PROG DIMINUE SA ZONE DE TIR  *
' *  D'UN CARRE DE 3*3 CASES         *
' *************************************
PROCEDURE dec_zone_tir(l%,c%)
LOCAL l0%,c0%
'
FOR l0%=l%-1 TO l%+1
FOR c0%=c%-1 TO c%+1
IF l0%>0 AND l0%<11 AND c0%>0 AND c0%<11
tir_ordinateur%(l0%,c0%)=1
ENDIF
NEXT c0%
NEXT l0%
RETURN
'
' ________________________________________
' ****************************************
' *  LE PROGRAMME NOTE LES CASES OU IL  *
' *  NE DOIT PLUS FAIRE DE TIRS         *
' ****************************************
PROCEDURE notation_cases
LOCAL l%,c%
'
FOR i%=1 TO dommages_navire%
l%=pos_navire_en_vue%(i%,1)
c%=pos_navire_en_vue%(i%,2)
@dec_zone_tir(l%,c%)
NEXT i%
RETURN
'
' ________________________________________________
' ************************************************
' *  LE PROG MARQUE QU'IL NE PEUT TIRER          *
' *  SUR LES CASES VOISINES DU SOUS-MARIN        *
' ************************************************
PROCEDURE marquage_sous_marin(l%,c%)
LOCAL l0%,c0%
'
FOR l0%=l%-1 TO l%+1
FOR c0%=c%-1 TO c%+1
IF l0%>0 AND l0%<11 AND c0%>0 AND c0%<11
tir_ordinateur%(l0%,c0%)=1
ENDIF
NEXT c0%
NEXT l0%
RETURN
'
' __________________________________
' **********************************
' *  TIR VERS LE HAUT DE L'ECRAN  *
' **********************************
> PROCEDURE tir_haut
LOCAL navire%
'
tir_programme%=0
DEC l_tir%
' * TEST SI DEPASSEMENT LIMITE ECRAN
IF l_tir%<1
@new_direction_tir
' * TEST SI TIR DEJA EFFECTUE DANS CETTE CASE
ELSE IF tir_ordinateur%(l_tir%,c_tir%)<>0
@new_direction_tir
ELSE
tir_programme%=1   ! LE PRG NOTE QU'IL A TIRE
navire%=grille_joueur%(l_tir%,c_tir%)
IF navire%<>navire_en_vue%
' * TIR DANS L'EAU
tir_ordinateur%(l_tir%,c_tir%)=1
HIDEM
@aff_tir_eau(l_tir%,c_tir%)
SHOWM
@new_direction_tir
ELSE
' * TIR SUR UNE CIBLE
INC dommages_navire%
@memorise_case_ennemi(l_tir%,c_tir%)
HIDEM
@impact_joueur(l_tir%,c_tir%)
```

```
SHOWM
ENDIF
ENDIF
RETURN
'
' ______________________________
' ******************************
' *  TIR VERS LE BAS DE L'ECRAN  *
' ******************************
> PROCEDURE tir_bas
LOCAL navire%
'
tir_programme%=0
INC l_tir%
' *  TEST SI DEPASSEMENT LIMITE ECRAN
IF l_tir%>10
@new_direction_tir
' *  TEST SI TIR DEJA EFFECTUE DANS CETTE CASE
ELSE IF tir_ordinateur%(l_tir%,c_tir%)<>0
@new_direction_tir
ELSE
tir_programme%=1   ! LE PRG NOTE QU'IL A TIRE
navire%=grille_joueur%(l_tir%,c_tir%)
IF navire%<>navire_en_vue%
' *  TIR DANS L'EAU
tir_ordinateur%(l_tir%,c_tir%)=1
HIDEM
@aff_tir_eau(l_tir%,c_tir%)
SHOWM
@new_direction_tir
ELSE
' * TIR SUR UNE CIBLE
INC dommages_navire%
@memorise_case_ennemi(l_tir%,c_tir%)
HIDEM
@impact_joueur(l_tir%,c_tir%)
SHOWM
ENDIF
ENDIF
RETURN
'
' ________________________________
' **********************************
' *  TIR VERS LA GAUCHE DE L'ECRAN  *
' **********************************
> PROCEDURE tir_gauche
LOCAL navire%
'
tir_programme%=0
DEC c_tir%
' *  TEST SI DEPASSEMENT LIMITE ECRAN
IF c_tir%<1
@new_direction_tir
' *  TEST SI TIR DEJA EFFECTUE DANS CETTE CASE
ELSE IF tir_ordinateur%(l_tir%,c_tir%)<>0
@new_direction_tir
ELSE
tir_programme%=1   ! LE PRG NOTE QU'IL A TIRE
navire%=grille_joueur%(l_tir%,c_tir%)
IF navire%<>navire_en_vue%
' *  TIR DANS L'EAU
tir_ordinateur%(l_tir%,c_tir%)=1
HIDEM
@aff_tir_eau(l_tir%,c_tir%)
SHOWM
@new_direction_tir
ELSE
' * TIR SUR UNE CIBLE
INC dommages_navire%
@memorise_case_ennemi(l_tir%,c_tir%)
HIDEM
@impact_joueur(l_tir%,c_tir%)
SHOWM
ENDIF
ENDIF
RETURN
'
' ________________________________
' **********************************
' *  TIR VERS LA DROITE DE L'ECRAN  *
' **********************************
> PROCEDURE tir_droit
LOCAL navire%
'
tir_programme%=0
INC c_tir%
' *  TEST SI DEPASSEMENT LIMITE ECRAN
IF c_tir%>10
@new_direction_tir
' *  TEST SI TIR DEJA EFFECTUE DANS CETTE CASE
ELSE IF tir_ordinateur%(l_tir%,c_tir%)<>0
@new_direction_tir
ELSE
tir_programme%=1   ! LE PRG NOTE QU'IL A TIRE
navire%=grille_joueur%(l_tir%,c_tir%)
IF navire%<>navire_en_vue%
' *  TIR DANS L'EAU
tir_ordinateur%(l_tir%,c_tir%)=1
HIDEM
@aff_tir_eau(l_tir%,c_tir%)
SHOWM
@new_direction_tir
ELSE
' * TIR SUR UNE CIBLE
INC dommages_navire%
@memorise_case_ennemi(l_tir%,c_tir%)
HIDEM
@impact_joueur(l_tir%,c_tir%)
SHOWM
ENDIF
```

```
ENDIF
RETURN
'
' ______________________________________________
' ***********************************************
' *  LE PROGRAMME DETERMINE UNE DIRECTION DE TIR  *
' ***********************************************
> PROCEDURE new_direction_tir
LOCAL d%
'
DO
d%=RANDOM(4)+1
EXIT IF test_direction_tir%(d%)=0
LOOP
test_direction_tir%(d%)=1
cible_direction%=d%
l_tir%=l_cible%
c_tir%=c_cible%
RETURN
'
' ________________________________
' *************************************
' *  L'ORDINATEUR TIRE SUR UN NAVIRE  *
' *  DEJA ENDOMMAGE                    *
' *************************************
> PROCEDURE tir_sur_navire
LOCAL taille%
'
IF cible_direction%=0
ARRAYFILL test_direction_tir%(),0
@new_direction_tir
ENDIF
'
DO
SELECT cible_direction%
CASE 1
@tir_gauche            ! TIR VERS LA GAUCHE
CASE 2
@tir_droit             ! TIR VERS LA DROITE
CASE 3
@tir_haut              ! TIR VERS LE HAUT
CASE 4
@tir_bas               ! TIR VERS LE BAS
ENDSELECT
EXIT IF tir_programme%=1
LOOP
'
' *  TEST SI LE NAVIRE EST COULE  *
taille%=@taille_navire(navire_en_vue%)
IF taille%=dommages_navire%
@notation_cases
INC pertes_joueur%
navire_en_vue%=0
dommages_navire%=0
cible_direction%=0
ENDIF
RETURN
'
' ****************************
' *  TIR DE L'ORDINATEUR     *
' *  SUR UNE CASE ALEATOIRE  *
' ****************************
PROCEDURE tir_aleatoire
LOCAL l%,c%,navire%
'
DO
l%=RANDOM(10)+1
c%=RANDOM(10)+1
EXIT IF tir_ordinateur%(l%,c%)=0
LOOP
navire%=grille_joueur%(l%,c%)
SELECT navire%
CASE 0
tir_ordinateur%(l%,c%)=1
HIDEM
@aff_tir_eau(l%,c%)
SHOWM
CASE sous_marin%
tir_ordinateur%(l%,c%)=1
INC pertes_joueur%
HIDEM
@impact_joueur(l%,c%)
SHOWM
@marquage_sous_marin(l%,c%)  ! MARQUAGES CASES
ADJACENTES
CASE torpilleur%,croiseur%,porte_avion%
navire_en_vue%=navire%
l_cible%=l%
c_cible%=c%
cible_direction%=0
dommages_navire%=1
@memorise_case_ennemi(l%,c%)
HIDEM
@impact_joueur(l%,c%)
SHOWM
ENDSELECT
RETURN
'
' *************************************
' *  GESTION DU TIR DE L'ORDINATEUR  *
' *************************************
> PROCEDURE gestion_ordinateur
IF navire_en_vue%=0
@tir_aleatoire
ELSE
@tir_sur_navire
ENDIF
IF pertes_joueur%=10
ordinateur_gagnant%=1
ENDIF
```

```
RETURN
'
' ****************************************
' *  IDENTIFICATION DE LA ZONE DE JEU  *
' *  SELECTIONNEE PAR LE JOUEUR       *
' ****************************************
FUNCTION identif_zone(xm%,ym%)
LOCAL rep%
'
rep%=0
IF @tstzone(xm%,ym%,172,28,120,120)
rep%=1
ENDIF
IF @tstzone(xm%,ym%,189,159,126,30)
rep%=2
ENDIF
IF @tstzone(xm%,ym%,6,159,126,30)
rep%=3
ENDIF
IF @tstzone(xm%,ym%,138,167,45,25)
rep%=4
ENDIF
RETURN rep%
ENDFUNC
'
' ___________________________________________
' *******************************************
' *  TEST SI LE JOUEUR VEUT QUITTER LE JEU  *
' *******************************************
> PROCEDURE tst_quitter_jeu
LOCAL m$,b%
'
m$="Voulez-vous vraiment|quittez ce jeu ?"
ALERT 0,m$,2,"Oui|Non",b%
IF b%=1
fin_prg%=1
ENDIF
RETURN
'
' ___________________
' *******************
' *  OPTION "INFOS"  *
' *******************
> PROCEDURE infos
LOCAL m$,b%
'
m$="BATAILLE NAVALE ATAMAG"
m$=m$+"|(C) 1992 Patrick Leclercq "
m$=m$+"|(C) 1992 Atari Magazine"
m$=m$+"|Gfa Basic 3.xx"
ALERT 0,m$,1,"Oui",b%
RETURN
'
'
' ***********************************
' *  EXECUTION DES FONCTIONS DU JEU  *
' ***********************************
PROCEDURE exec_commande(c%)
SELECT c%
CASE 2
nouvelle_partie%=1
CASE 3
@tst_quitter_jeu
CASE 4
@infos
ENDSELECT
RETURN
'
' *********************************
' *  LECTURE DU TYPE D'UN NAVIRE  *
' *********************************
> FUNCTION get_type_navire(n%)
LOCAL rep%
'
rep%=-1
SELECT n%
CASE 0
rep%=0
CASE 1
rep%=porte_avion%
CASE 2,3
rep%=croiseur%
CASE 4,5,6
rep%=torpilleur%
CASE 7,8,9,10
rep%=sous_marin%
ENDSELECT
RETURN rep%
ENDFUNC
'
' ********************************
' *  IDENTIFICATION DE LA CASE    *
' *  SELECTIONNEE PAR LE JOUEUR   *
' *  VALABLE UNIQUEMENT POUR LA   *
' *  GRILLE DE TIR DU JOUEUR      *
' ********************************
> PROCEDURE identif_case_tir(xm%,ym%,VAR l%,c%)
LOCAL px%,py%
'
px%=xm%-172
py%=ym%-28
l%=(py%/ty_case%)+1
c%=(px%/tx_case%)+1
RETURN
'
' ********************************
' *  AFFICHAGE D'UNE CROIX VIDE  *
' ********************************
> PROCEDURE aff_croix_vide(l%,c%)
LOCAL xc%,yc%
```

```
'
xc%=(c%-1)*tx_case%+172
yc%=(l%-1)*ty_case%+28
COLOR noir
DEFLINE ,1
LINE xc%+3,yc%+3,xc%+9,yc%+9
LINE xc%+3,yc%+9,xc%+9,yc%+3
RETURN
'
' ******************************
' *  AFFICHAGE D'UN TIR REUSSI  *
' ******************************
> PROCEDURE tir_reussi(l%,c%)
LOCAL xc%,yc%
'
xc%=(c%-1)*tx_case%+172
yc%=(l%-1)*ty_case%+28
COLOR rouge
DEFFILL rouge
PCIRCLE xc%+6,yc%+6,4
DEFLINE ,3
LINE xc%,yc%,xc%+11,yc%+11
LINE xc%,yc%+11,xc%+11,yc%
RETURN
'
' ***************************
' *  AFFICHAGE D'UN IMPACT  *
' ***************************
> PROCEDURE aff_impact(l%,c%)
LOCAL xc%,yc%
'
xc%=(c%-1)*tx_case%+172
yc%=(l%-1)*ty_case%+28
DEFFILL rouge
PCIRCLE xc%+6,yc%+6,4
RETURN
'
' *********************************
' *  AFFICHAGE D'UN NAVIRE COULE  *
' *********************************
> PROCEDURE aff_navire_coule(n%)
LOCAL l%,c%
'
FOR i%=1 TO nb_impacts%(n%)
l%=pos_impact%(n%,i%,1)
c%=pos_impact%(n%,i%,2)
@tir_reussi(l%,c%)
NEXT i%
RETURN
'
' *******************************************
' *  MEMORISATION DE LA POSITION DE L'IMPACT  *
' *******************************************
> PROCEDURE memorisation_impact(n%,l%,c%)
LOCAL impact%
'
impact%=nb_impacts%(n%)
pos_impact%(n%,impact%,1)=l%
pos_impact%(n%,impact%,2)=c%
RETURN
'
' ******************************
' *  GESTION DU TIR DU JOUEUR  *
' ******************************
> PROCEDURE tir_joueur(xm%,ym%)
LOCAL l%,c%,navire%,type_navire%,taille%
'
@identif_case_tir(xm%,ym%,l%,c%)
navire%=grille_ordinateur%(l%,c%)
type_navire%=@get_type_navire(navire%)
SELECT type_navire%
CASE 0  ! TIR DANS L'EAU
grille_ordinateur%(l%,c%)=-10
HIDEM
@aff_croix_vide(l%,c%)
SHOWM
tir_joueur%=1
CASE sous_marin%,torpilleur%,croiseur%,porte_avion%
HIDEM
@aff_impact(l%,c%)
' MEMORISATION TIR
grille_ordinateur%(l%,c%)=-navire%
INC nb_impacts%(navire%)
@memorisation_impact(navire%,l%,c%)
' TEST SI NAVIRE COULE
taille%=@taille_navire(type_navire%)
IF nb_impacts%(navire%)=taille%
@aff_navire_coule(navire%)
INC pertes_ordinateur%
' * TEST SI LA FLOTTE ORDINATEUR EST DETRUITE
IF pertes_ordinateur%=10
joueur_gagnant%=1
ENDIF
ENDIF
tir_joueur%=1
SHOWM
ENDSELECT
RETURN
'
' ***********************************
' *  GESTION DES ACTIONS DU JOUEUR  *
' ***********************************
PROCEDURE gestion_joueur
LOCAL xm%,ym%,km%,zone%
'
tir_joueur%=0
DO
@att_clic(xm%,ym%,km%)
zone%=@identif_zone(xm%,ym%)
SELECT zone%
```

```
CASE 2,3,4,5,6
@exec_commande(zone%)
CASE 1
@tir_joueur(xm%,ym%)
ENDSELECT
' TEST SI LE JOUEUR A TIRE
EXIT IF tir_joueur%=1
' TEST SI SORTIE DU PROG
EXIT IF fin_prg%=1
EXIT IF nouvelle_partie%=1
LOOP
RETURN
'
' ********************
' *  GESTION DU JEU  *
' ********************
PROCEDURE gestion_jeu
HIDEM
@aff_jeu
SHOWM
fin_prg%=0
DO
@gestion_joueur     ! GESTION ACTIONS JOUEUR
EXIT IF fin_prg%=1
@gestion_ordinateur ! GESTION ACTIONS ORDINATEUR
EXIT IF fin_prg%=1
' * TEST SI JOUEUR GAGNANT
EXIT IF joueur_gagnant%=1
' *  TEST SI ORDINATEUR GAGNANT
EXIT IF ordinateur_gagnant%=1
' * TEST SI NOUVELLE PARTIE
EXIT IF nouvelle_partie%=1
LOOP
RETURN
'
' *****************************************
' *  TEST SI UNE ZONE DE LA GRILLE DE LA *
' *  FLOTTE DE L'ORDINATEUR EST VIDE.     *
' *****************************************
' *  l%,c%: position de la zone à tester  *
' *  nb_l%,nb_c%: dimension de la zone    *
' *****************************************
> FUNCTION tst_grille_ordinateur(l%,c%,nb_l%,nb_c%)
LOCAL l0%,c0%,rep%
'
rep%=TRUE
FOR l0%=l% TO l%+nb_l%-1
FOR c0%=c% TO c%+nb_c%-1
IF (c0%>0) AND (c0%<11) AND (l0%>0) AND (l0%<11)
IF grille_ordinateur%(l0%,c0%)<>0
rep%=FALSE
ENDIF
ENDIF
NEXT c0%
NEXT l0%
'
RETURN rep%
ENDFUNC
'
' *******************************************
' *  TEST SI UN NAVIRE PEUT ETRE           *
' *  ECRIT DANS LA GRILLE DE L'ORDINATEUR *
' *******************************************
> FUNCTION tst_navire_ordinateur(l%,c%,n%,d%)
LOCAL rep%,t%
'
rep%=TRUE
t%=@taille_navire(n%)
' ***********************
' *  NAVIRE HORIZONTAL  *
' ***********************
IF d%=1
IF c%>(11-t%) ! TEST SI DEPASSEMENT GRILLE
rep%=FALSE
ENDIF
' *  TEST SI NAVIRE DANS CASES ADJACENTES  *
IF NOT (@tst_grille_ordinateur(l%-1,c%-1,3,t%+2))
rep%=FALSE
ENDIF
ELSE
' *********************
' *  NAVIRE VERTICAL  *
' *********************
IF l%>(11-t%) ! TEST SI DEPASSEMENT GRILLE
rep%=FALSE
ENDIF
' *  TEST SI NAVIRE DANS CASES ADJACENTES  *
IF NOT (@tst_grille_ordinateur(l%-1,c%-1,t%+2,3))
rep%=FALSE
ENDIF
ENDIF
'
RETURN rep%
ENDFUNC
'
' ************************************
' *  STOCKAGE D'UN NAVIRE EN MEMOIRE *
' *  DANS LA GRILLE DE L'ORDINATEUR   *
' ************************************
' *  l%,c%: position du navire       *
' *  n%:    numéro du navire          *
' *  d%:    direction du navire       *
' ************************************
> PROCEDURE stockage_navire_ordinateur(l%,c%,n%,d%)
LOCAL l0%,c0%,type%,taille%
'
type%=@get_type_navire(n%)
taille%=@taille_navire(type%)
IF d%=1                   ! NAVIRE HORIZONTAL
FOR c0%=c% TO c%+taille%-1
```

```
grille_ordinateur%(l%,c0%)=n%
NEXT c0%
ELSE                    ! NAVIRE VERTICAL
FOR l0%=l% TO l%+taille%-1
grille_ordinateur%(l0%,c%)=n%
NEXT l0%
ENDIF
RETURN
'
' *************************************
' *  PLACEMENT FLOTTE DE L'ORDINATEUR  *
' *************************************
> PROCEDURE placement_flotte_ordinateur
LOCAL l%,c%,d%,n%,i%,type%
'
' *  PLACEMENT PORTE-AVION
type%=porte_avion%
n%=1
DO
l%=RANDOM(10)+1
c%=RANDOM(10)+1
d%=RANDOM(2)+1
EXIT IF @tst_navire_ordinateur(l%,c%,type%,d%)
LOOP
@stockage_navire_ordinateur(l%,c%,n%,d%)
'
' PLACEMENT CROISEURS
type%=croiseur%
n%=2
FOR i%=1 TO 2
DO
l%=RANDOM(10)+1
c%=RANDOM(10)+1
d%=RANDOM(2)+1
EXIT IF @tst_navire_ordinateur(l%,c%,type%,d%)
LOOP
@stockage_navire_ordinateur(l%,c%,n%,d%)
INC n%
NEXT i%
'
' PLACEMENT TORPILLEURS
type%=torpilleur%
n%=4
FOR i%=1 TO 3
DO
l%=RANDOM(10)+1
c%=RANDOM(10)+1
d%=RANDOM(2)+1
EXIT IF @tst_navire_ordinateur(l%,c%,type%,d%)
LOOP
@stockage_navire_ordinateur(l%,c%,n%,d%)
INC n%
NEXT i%
'
' PLACEMENT SOUS-MARINS
type%=sous_marin%
n%=7
FOR i%=1 TO 4
DO
l%=RANDOM(10)+1
c%=RANDOM(10)+1
d%=RANDOM(2)+1
EXIT IF @tst_navire_ordinateur(l%,c%,type%,d%)
LOOP
@stockage_navire_ordinateur(l%,c%,n%,d%)
INC n%
NEXT i%
RETURN
'
'
' *********************************************
' *  INITIALISATION DES DONNEES DU PROGRAMME  *
' *********************************************
PROCEDURE init_prg
' ****************************
' *  DEFINITION DES COULEURS  *
' ****************************
blanc=1
noir=0
rouge=2
vert=3
bleu2=10
bleu3=5
marron=6
vert2=7
gris=8
gris2=9
bleu=10
bleu4=11
violet=12
violet2=13
jaune2=14
jaune=15
'
ARRAYFILL grille_joueur%(),0
ARRAYFILL grille_ordinateur%(),0
ARRAYFILL tir_ordinateur%(),0
'
' ********************************
' *  NB DE NAVIRES DISPONIBLES  *
' ********************************
navires_disponibles%(sous_marin%)=4
navires_disponibles%(torpilleur%)=3
navires_disponibles%(croiseur%)=2
navires_disponibles%(porte_avion%)=1
'
' *********************************************
' *  TABLEAU CONTENANT LES POSITIONS DES NAVIRES  *
' *  DE L'ORDINATEUR TOUCHE PAR LE JOUEUR         *
' *  pos_impact%(n,i,p)                           *
```

```
' *  n = numéro de navire (1 à 10)                    *
' *  i = numéro impact (1 à 4)                        *
' *  p = ligne ou colonne (1=ligne; 2=colonne)        *
' **************************************************
ARRAYFILL nb_impacts%(),0
ARRAYFILL pos_impact%(),0
'
' **************************
' *  VARIABLE GLOBALE      *
' *  DIRECTION DES NAVIRES *
' *  1 = horizontale       *
' *  2 = verticale         *
' **************************
navire_direction%=1
'
' ************************************************
' *  VARIABLE GLOBALE                            *
' *  TYPE DE NAVIRE COURANT LORS DE LA CREATION  *
' *  0 = pas de navire                           *
' ************************************************
type_navire%=sous_marin%
'
' ***********************
' *  VARIABLE FIN DE PRG  *
' *  0 = Continuer prg    *
' *  1 = Quitter prg      *
' ***********************
fin_prg%=0
'
' *************************************
' *  VARIABLES CONTENANT LE VAINQUEUR  *
' *************************************
joueur_gagnant%=0
ordinateur_gagnant%=0
'
nouvelle_partie%=0
'
' *********************************
' *  ETAT DES FLOTTES EN PRESENCE  *
' *********************************
pertes_ordinateur%=0
pertes_joueur%=0
'
' *****************************************
' *  VARIABLE INDIQUANT QUE L'ORDINATEUR  *
' *  A REPERE UNE CIBLE IMPORTANTE        *
' *  (TORPILLEUR, CROISEUR OU PORTE_AVION) *
' *  0    ==> Pas de cible importante     *
' *  <>0 ==> Type de cible                *
' *****************************************
navire_en_vue%=0
'
' *********************************
' *  POSITION DE LA CIBLE REPEREE  *
' *********************************
l_cible%=0
c_cible%=0
'
' **************************
' *  DIRECTION DE LA CIBLE  *
' *  0= pas connue          *
' **************************
cible_direction%=0
'
' ****************************************
' *  DOMMAGES SUBIS PAR LE NAVIRE SOUMIS  *
' *  AU TIR DU PROGRAMME                  *
' ****************************************
dommages_navire%=0
'
' *************************************
' *  VARIABLE INDIQUANT QUE LE JOUEUR  *
' *  A TIRE SUR LA FLOTTE ENNEMI.      *
' *************************************
tir_joueur%=0
'
' ***************************
' *  DEFINITION CASE GRILLE  *
' ***************************
tx_case%=12
ty_case%=12
'
VSETCOLOR 0,0
VSETCOLOR 1,7,7,7
RETURN
'
' ****************************
' *  RESTAURATION DES COULEURS  *
' *  DU BUREAU GEM              *
' ****************************
> PROCEDURE reset_prg
VSETCOLOR 1,0
VSETCOLOR 0,7,7,7
RETURN
'
' ___________________________
' *********************
' *  EXECUTION DU JEU  *
' *********************
PROCEDURE exec_jeu
LOCAL m$,b%
'
DO
@init_prg
@dessin_ecr_pos
SHOWM
@ges_creation
@placement_flotte_ordinateur
@gestion_jeu
IF joueur_gagnant%=1
```

```
m$="VOUS AVEZ GAGNE|Une autre partie ? "
ALERT 0,m$,1,"Oui|Non",b%
IF b%=2
fin_prg%=1
ENDIF
ENDIF
IF ordinateur_gagnant%=1
m$="VOUS AVEZ PERDU|Une autre partie ? "
ALERT 0,m$,1,"Oui|Non",b%
IF b%=2
fin_prg%=1
ENDIF
ENDIF
EXIT IF fin_prg%=1
LOOP
RETURN
'
' __________________________
' ***************************
' *  PROCEDURE PRINCIPALE  *
' ***************************
PROCEDURE main
RANDOMIZE
GRAPHMODE 2
@exec_jeu
@reset_prg
RETURN
```

PROGRAMMATION EN GFA BASIC
SOS GFA Basic

Le nombre de questions que se posent les utilisateurs concernant le GFA Basic va croissant. Face à cet afflux, voici une nouvelle rubrique destinée à répondre aux questions types et à résoudre les problèmes les plus courants.

La liste des questions ne se veut pas exhaustive. Nous essayons de pallier les problèmes les plus couramment rencontrés par les utilisateurs. Envoyez vos questions par courrier à l'adresse du magazine ou tapez 3615 Atari bal GFA Basic.

Je possède la version 3.03 du GFA Basic et j'aimerais savoir si cela vaut la peine d'acheter une version plus récente?

C'est une question difficile dont la réponse dépend beaucoup de votre budget puisque le prix d'une mise à jour est de quelques centaines de francs. La dernière version du *GFAGFA Basic (3.5E)* n'est pas fondamentalement différente de la version 3.03. Elle possède quelques instructions spécifiques permettant de gérer les particularités du STE (manettes de jeu, fonctions sonores et graphiques), mais ces fonctions peuvent facilement être programmées en *GFA Basic 3.03* en utilisant les fonctions XBIOS. De plus, les matheux seront heureux de savoir que les versions 3.5 et 3.5E du *GFAGFA Basic* possèdent des fonctions matriciels très puissantes. Les auteurs de la version 3.5E ont corrigé bon nombre de bogues présents dans les versions précédentes où il arrivait parfois qu'un programme ne fonctionne plus une fois compilé.

Le GFA Basic permet-il d'écrire des logiciels équivalents aux logiciels vendus dans le commerce?

La réponse à votre question est oui et non. Oui, parce que certains des logiciels du commerce sont effectivement écrits en *GFA Basic*, notamment des jeux d'aventures et des petits programmes de gestion. Non, car le *Basic* est un langage d'initiation conçu pour les débutants en informatique. Le *GFA Basic* est une version très évoluée du *Basic* de base, mais reste toujours un langage à la structure simple manquant de mécanismes élaborés pour gérer des problèmes complexes. Il est possible d'écrire de nombreux petits programmes en *GFA*, mais pour des applications lourdes comme un traitement de textes, une gestion de base de données, un logiciel d'analyse statistique ou un logiciel de PAO, il faut absolument un langage professionnel de type *Pascal* ou *C*. Il est possible d'écrire n'importe quel programme en *Basic*, mais comme ce langage n'est prévue que pour écrire des choses simples, ce serait un véritable enfer pour le programmeur qui ferait des bricolages monstrueux et mettrait énormément de temps pour mettre au point un programme peu fiable. Un programmeur amateur qui désire écrire un gros programme avec beaucoup de données gagnera du temps en apprenant un langage évolué du type *Pascal* plutôt que de commencer à écrire directement son application en *GFA Basic*. La majeure partie des logiciels professionnels ST est programmée en langage *C* avec, éventuellement, quelques routines critiques écrites en *Assembleur*.

Existe-t-il un logiciel permettant de décompiler un programme GFA Basic (non protégé) afin de le lister pour effectuer des corrections? S'agit-il d'un désassembleur?

A notre connaissance, il n'existe aucun utilitaire permettant de génerer un listing en *GFA Basic* à partir d'un programme compilé. C'est possible en théorie, mais cela générerait un code difficile à relire puisque le programme compilé ne conserve ni le nom des procédures, ni celui des variables. De plus, comme le compilateur procède parfois à des optimisations de codes, certaines portions du programmes risquent de ne pas beaucoup ressembler au listing initial. Il est possible d'utiliser un désassembleur pour étudier un programme compilé, mais le listing produit est en assembleur et non en *GFA Basic*. Cela peut être utile pour un programmeur en langage machine, mais pas pour un utilisateur du *Basic*. De plus, interrogés sur la question, certains spécialistes de l'assembleur ont expliqué que le code machine généré par le compilateur du ST n'était pas très propre et donc difficilement réutilisable.

J'ai tout particulièrement apprécié vos articles sur la création des jeux de rôle bien que l'éditeur de cartes style Ultima soit un peu trop complexe pour moi. Envisagez-vous d'écrire d'autres articles dans le même style, sur la création des jeux de rôle ou d'autres types de jeux?

En effet, dans les numéros à venir, nous envisageons d'écrire des articles sur la manière de programmer certains types de logiciels du commerce. Pour des raisons de place et de temps, ces articles ne seront pas accompagnés de programmes fonctionnels, mais présenteront diverses techniques que vous pourrez mettre en pratique dans vos propres programmes. Il y aura certainement

d'autres articles sur les jeux de rôle.

J'ai écrit un programme de gestion de fichiers et mes routines de sauvegarde disque fonctionnent mal.
Qu'en pensez-vous?
Voici le type même de question que nous préférons ne pas recevoir. Non seulement, il nous est impossible de déboguer tous vos programmes par manque de temps, mais de plus la question est complètement flou. Si vous voulez poser des questions sur un problème précis, donnez-nous le maximum d'informations sur ce qui devrait se passer et sur ce qui se passe réellement, ainsi qu'une partie de votre listing avec une explication détaillée sur son fonctionnement.

J'ai écrit un logiciel de gestion de données en GFA Basic exploitant un fichier article d'environ 100 Ko. Lors d'un transfert de fichiers dans un Ram Disque (sur C: avec FLEXDISC), j'ai eu l'idée de modifier le programme pour la lecture sur C: et j'ai remarqué que le temps d'accès est divisé par 2 et plus. Existe-t-il un moyen pour charger ce fichier en mémoire, soit directement, soit à l'aide d'un petit programme en GFA (et avec quelle instruction) avant d'exécuter mon programme principal?
Les Ram Disques fonctionnement beaucoup plus vite que les disquettes et même que les disques durs, car ils ne possèdent aucune pièce mécanique ralentissant le transfert des données. Les seules choses que l'on puisse leur reprocher, c'est qu'ils prennent beaucoup de place mémoire et que leur contenu s'efface si l'alimentation du ST est coupée, à la différence des disquettes et des disques durs qui conservent en permanence leurs données. Vous pouvez copier vos fichiers dans votre Ram Disque en utilisant le bureau GEM, ou en utilisant une routine comme la procédure:
`copie_fichier(nom1$,nom2$)`.
Cette dernière n'est pas très rapide à l'exécution.
Vous pouvez l'améliorer en effectuant non pas la copie octet par octet, mais bloc d'octets par bloc d'octets (voir listing 1).

Je voudrais afficher constamment la valeur d'une variable dans un coin de l'écran, mais l'affichage se fait sur chaque ligne et finit par détruire mon image. Que dois-je faire?
C'est un difficulté simple à résoudre, mais qui pose toujours beaucoup de problèmes aux débutants. L'instruction PRINT permet d'afficher une information quelconque sur l'écran. L'instruction LOCATE précise à quelle ligne et à quelle colonne de l'écran s'effectue l'affichage.

```
LOCATE 1,1
PRINT var
```

L'instruction PRINT n'affiche que le nombre de caractères nécessaire à l'affichage de la variable devant être affichée. Cela peut poser un problème si un nombre important est déjà affiché à l'écran.
Prenons le cas où le programme affiche la valeur 56 à un emplacement de l'écran où se trouve déjà écrit la valeur 34. Les caractères 56 vont effacer les caractères 34. En revanche, si vous tentez d'afficher la valeur 56 alors qu'il y a déjà sur l'écran la valeur 234, les deux premiers caractères de l'écran seront remplacés par 56, mais le troisième restera. L'écran contiendra alors la valeur 564. Pour éviter cet effet, il faut effacer la portion de l'écran où doit se faire l'affichage. Cette opération est exécutable à l'aide de l'instruction SPACE$ qui génère une chaîne alphanumérique constituée uniquement d'espaces (voir listing 2).

J'ai programmé une boucle FOR NEXT allant de 1 à 10 avec un pas de 2 et j'ai constaté que la variable i ne prend jamais la valeur 10. Est-ce un bogue du GFA Basic? Voici un programme qui prouve mes dires:

```
FOR i=1 TO 10 STEP 2
  PRINT i
NEXT i
```

Les chiffres affichés sont: 1, 3, 5, 7, 9
Ce n'est pas vraiment un bogue, mais plutôt une conséquence de la manière dont le *GFA Basic* gère les boucles FOR NEXT.
Lorsque le programme traite l'instruction NEXT, il incrémente la variable i et compare le résultat du calcul avec la valeur de fin de la boucle. Si la valeur de i est supérieure ou égale à la valeur de fin de boucle, la boucle FOR NEXT se termine.
Dans le cas présent, la boucle s'arrête sur la valeur 9, car 9+2=11, et 11 est supérieure ou égale à 10. Ceci est assez subtile et peut occasionner des bogues difficiles à trouver.
Il faut faire très attention chaque fois que vous utilisez des boucles FOR NEXT avec un pas d'incrémentation différent de 1.
Remarque: si le pas d'incrémentation d'une boucle est différent de 1, la valeur de sortie de la boucle n'est pas obligatoirement la valeur de fin de la boucle.
Cela peut être une source d'erreurs à éviter, comme le montre l'exemple suivant:

```
FOR i=1 TO 10 STEP 2
  PRINT i
NEXT i
PRINT "Valeur de sortie :";i
```

```
1
3
7
9
Valeur de sortie : 11
```

Quelle est la meilleure manière de rechercher une chaîne alphanumérique dans un tableau de caractères?
La technique la plus simple pour rechercher le numéro de stockage d'une chaîne dans un tableau alphanumérique est d'utiliser une boucle FOR NEXT afin de tester une à une les différentes chaînes du tableau. Le programme suivant recherche à quel emplacement du tableau t$() se trouve la chaîne «Melnibonné». La variable rep contient le résultat de la recherche, la valeur -1 indiquant que la recherche n'a pas abouti (voir listing 3).

Cette technique est simple à programmer, mais elle présente un gros inconvénient: même si la chaîne m$ se trouve au début du tableau, la routine explore la totalité de t$(), ce qui représente une perte de temps importante. Pour gagner du temps, il faut forcer la routine à cesser la recherche dés que la chaîne de recherche est trouvée. La manière la plus simple est d'écrire la valeur de fin de boucle, c'est-à-dire 100, dans l'indice de boucle, autrement dit la variable i. La boucle s'arrêtera alors automatiquement (voir listing 4).

La modification des indices de boucles est une technique dangereuse qu'il faut utiliser avec beaucoup de précaution. En effet, cela risque de créer des boucles sans fin qui ne s'arrêteront jamais comme le montre un peu caricaturalement l'exemple suivant:

```
FOR i=1 TO 100
  i=50
NEXT i
```

Comment faire pour afficher sur l'écran une image Degas Elite, avec sa bonne palette de couleurs?

Cette question revient souvent dans le courrier des lecteurs et sur le serveur 3615 Atari. Les images Degas sont stockées dans des fichiers binaires ayant la structure suivante: deux octets mémorisant la résolution de l'image (0=basse résolution; 1=moyenne résolution; 2=haute résolution), 32 octets pour la palette de couleurs de l'image et 32000 octets contenant l'image écran. La routine @charge_degas(nom$) affiche sur l'écran du ST l'image *Degas* contenue dans le fichier nom$. Elle utilise l'instruction BGET pour charger en mémoire une partie du fichier. Si vous voulez plus de renseignement sur le fonctionnement de cette routine, reportez-vous à l'article sur les fichiers binaires paru dans Atari Magazine n°16 (voir listing 5).

Quel est le rôle de l'instruction RESERVE et comment déterminer la valeur de son paramètre numérique?
Le *GFA Basic* stocke le contenu de ces variables dans une zone mémoire particulière. L'instruction RESERVE définit la taille de cette zone. Certains programmes ont besoin de beaucoup de mémoire, alors que d'autres se contentent de peu. Si un programme manque de mémoire, il s'interrompt et affiche une boîte d'alerte avec le message «mémoire pleine». Le programme suivant vous permettra de visualiser ce message d'erreur:

```
RESERVE 0
DIM t%(1000)
PRINT "Test RESERVE"
END
```

Le tableau t%() contient 1001 nombres (de 0 à 1000), occupant chacun 4 octets. L'instruction RESERVE doit donc allouer un peu plus de 4000 octets au *GFA Basic* pour que le programme puisse s'exécuter. Avec une valeur de 5000, vous prenez une bonne marge de sécurité.

```
RESERVE 5000
DIM t%(1000)
PRINT "Test RESERVE"
END
```

Pour déterminer la taille du paramètre de RESERVE, il faut tenir compte de toutes les variables du programme. Les variables qui prennent le plus de place sont les tableaux, les variables alphanumériques et les images. Une image écran stockée dans une variable alphanumérique avec l'instruction SGET occupe 32000 octets. Malgré son allocation mémoire de 50000 octets, le programme suivant ne fonctionne pas: voir listing 6.

Le stockage des deux images écran dans les variables alphanumériques img1$ et img2$ nécessite 2*32000 octets, soit 64000 octets. Il faut donc réserver au moins 64000 octets pour qu'il fonctionne correctement (voir listing 7).

Le *GFA Basic* gère sa mémoire de manière dynamique, c'est-à-dire que la taille de la zone mémoire utilisée varie au fur et à mesure des besoins du programme. Cela simplifie grandement la tâche du programmeur. Malheureusement, il est impossible de savoir à l'avance si le programme nécessite plus de mémoire qu'il ne lui en ait alloué avec l'instruction RESERVE. Le programme de l'exemple précédent ne plante pas lors du stockage en mémoire de la première image, mais pendant le stockage de la seconde image étant donné qu'il manque une douzaine de Ko. Il faut donc calculer la taille minimale de la mémoire avec précision. Un programme simple qui contient peu de variables se contente d'une allocation mémoire de 50000 octets.

Comment faire pour stocker plusieurs images en mémoire?
L'instruction SGET permet de stocker une image écran dans une variable alphanumérique. Pour mémoriser plusieurs images en mémoire, il faut utiliser plusieurs variables alphanumériques ou un tableau alphanumérique. L'exemple suivant charge plusieurs images en mémoire et les affiche rapidement les unes à la suite des autres. L'instruction RESERVE alloue 200000 octets au *GFA Basic*, ce qui suffit très largement pour stocker les 3 images et les variables du programme. La procédure @charge_degas(nom$), qui affiche une image *Degas* sur l'écran du ST, est décrite au début de cet article (voir listing 8).

Attention: ce petit programme ne conserve pas les palettes de couleurs des différentes images. Pour que l'affichage soit correct, il faut donc que les images aient toutes la même palette.

J'ai écrit un petit programme de dessin avec la version 3.03 de l'interpréteur du GFA Basic, puis je l'ai compilé pour aller plus vite. La version compilée marche aussi bien que la version interprétée, mais au début du programme le curseur souris a la forme d'une abeille. Est-ce que mon compilateur fonctionne correctement?
Rassurez-vous, votre compilateur fonctionne correctement, bien qu'il soit peut-être temps d'acheter la mise à jour du *GFA Basic 3.5E*. Ce qui se passe, c'est que les programmes écrits en *GFA Basic* ne changent pas la forme de la souris. Si vous lancez un programme à partir de l'interpréteur, il n'y a pas de problème puisque la forme de la souris est déjà celle d'une flèche. En revanche, si vous cliquez sur l'icône d'un programme *GFA* compilé, le bureau GEM change la souris en abeille pour montrer qu'il exécute une tâche et lance l'exécution du programme *GFA*. La forme de la souris reste toujours celle d'une abeille. Pour avoir un curseur souris en forme de flèche, il faut toujours commencer les programmes par un appel à l'instruction DEFMOUSE 0 qui change la forme de la souris en une flèche. La liste suivante vous montre les formes de souris disponibles dans la ROM du ST:

0 : Flèche
1 : Double parenthèse
2 : Abeille
3 : Main avec index pointé
4 : Main ouverte
5 : Réticule mince
6 : Réticule épais
7 : Réticule encadré

De la même manière que GEM, vos programmes peuvent indiquer qu'ils exécutent une tâche en changeant la forme de la souris par une abeille. Pour ce faire, il faut utiliser DEFMOUSE deux fois comme le montre la routine suivante: voir listing 9.

Le GFA basic possède de nombreuses structures de contrôle permettant de simplifier l'écriture des programmes, mais il y en a tellement que je m'y perds. Je ne parviens pas à trouver la boucle appropriée à chaque problème. Comment choisir entre les boucles DO WHILE, DO UNTIL, LOOP WHILE, LOOP UNTIL, DO LOOP, REPEAT UNTIL et WHILE WEND?
Les auteurs du *GFA Basic* ont voulu faire un «coup de zèle» en intégrant tous ces types de boucle dans leur langage, mais ils n'ont réussi qu'à embrouiller l'esprit des utilisateurs. Les développeurs professionnels qui programment toute la journée n'utilisent qu'un ou deux types de boucle. Toutes ces

boucles ont la même fonction, à savoir répéter une séquence d'instructions jusqu'à ce qu'une condition de sortie apparaisse. Elles diffèrent sur la manière et l'endroit dont la condition de sortie doit être testée. Afin de vous simplifier la vie, n'utilisez que la boucle DO LOOP. Cette dernière répète une séquence d'instructions tant que la condition de sortie présente dans l'instruction EXIT IF n'est pas réalisée. Le gros avantage de cette structure, c'est que le test de sortie de la boucle peut être placé n'importe où à l'intérieur de celle-ci. C'est très pratique pour de nombreuses applications. Le programme suivant vous montre un exemple pratique d'utilisation de la boucle DO LOOP. La boucle s'exécute tant que la variable alphanumérique ne contient pas le caractère [E], c'est-à-dire tant que l'utilisateur ne presse pas sur la touche [E].

```
n=0
DO
  INC n
  a$=INKEY$
  EXIT IF a$="E"
LOOP
```

Instituteur, j'ai, dans ma classe, formé un atelier atariste (3 ordinateurs), et j'ai également, pour l'école, constitué un programme pour établir nos carnets de notes que je voudrais perfectionner. Voilà mon problème: je cherche, mais en vain, à supprimer dans un fichier à accès direct le nom des enfants qui seraient partis en cours d'année. Pour ce faire, je copie d'abord le fichier A (par exemple) sur le fichier B. J'ai auparavant marqué d'un * les fichiers à supprimer dans A, et je cherche par une boucle à recopier les fichiers non marqués d'un *. Beaucoup d'heures passées: je commence à déprimer. Merci de me venir en aide, si c'est possible! Je joins à cette lettre la partie de mon programme qui me pose problème (voir listing 10).

Plusieurs «choses» peuvent provoquer un mauvais fonctionnement de votre routine. D'une part, vous appelez la routine de fermeture des fichiers à l'intérieur même de la boucle FOR NEXT, sans vous occuper d'arrêter la boucle. Cette dernière continue de s'exécuter et tente de lire des informations à partir d'un fichier fermé, provoquant le blocage du programme et l'apparition d'un message d'erreur. Etant donné que le programme ne connaît pas à l'avance le nombre d'enregistrements qu'il va lire dans le fichier, il ne faut pas utiliser une boucle FOR NEXT, mais une boucle DO LOOP dont la sortie serait contrôlée par une instruction EXIT IF. D'autre part, vous gérez très mal les numéros des enregistrements du fichier B. Puisque certains élèves du fichier A disparaissent, les numéros de référence du fichier B sont différents de ceux du premier fichier. Enfin, vous modifiez la valeur de la variable i%, compteur de boucle un peu avant la fin de la boucle. Il faut éviter de modifier la valeur d'une variable indice de boucle à l'intérieur même de la boucle, sous peine de graves problèmes.
Voici une routine qui répond à vos demandes, conçue autour d'une boucle DO LOOP et non d'une boucle FOR NEXT. La variable n% contient le compteur d'élèves du fichier A. La variable eleve% est le compteur d'élèves du fichier B. Lorsque le programme détecte la fin du fichier A, il recopie l'indicateur de fin de fichier dans le fichier B et active la condition de sortie, c'est-à-dire qu'il écrit la valeur 1 dans la variable sortie%. L'instruction EXIT IF indique au programme qu'il doit quitter la boucle DO LOOP pour exécuter les instructions situées après le LOOP. C'est là que doivent être placées les instructions de fermeture des fichiers à accès direct (voir listing 11).

Une dernière remarque: dans votre listing, si le nom de l'élève commence par le caractère *, le programme utilise l'instruction GOTO pour sauter à la fin de la boucle FOR NEXT. C'est une mauvaise programmation. Il ne faut jamais utiliser l'instruction GOTO, reliquat des premiers Basic. Cette maudite instruction ne sert qu'à créer des programmes non fiables et illisibles. Aucun des programmes publiés dans Atari Magazine depuis plus de deux ans ne contient le moindre GOTO. Pour supprimer cette horreur, au lieu de tester si la première lettre du nom est le caractère *, il suffit de faire le contraire et donc de tester si la première lettre du nom n'est pas le caractère *. Le même programme sans GOTO est plus simple et plus lisible, comme le montre l'exemple suivant: voir listing 12.

Qu'est-ce que le mode Superviseur? Pourquoi l'instruction SPOKE travaille en mode Superviseur et pas POKE?
Le microprocesseur 68000 qui équipe votre ST est un circuit électronique complexe capable de fonctionner dans deux modes différents: le mode utilisateur et le mode *superviseur*. Un programme fonctionnant mode superviseur peut écrire à n'importe quel endroit de la mémoire alors qu'un programme écrit dans le mode utilisateur ne peut accéder à certaines zones mémoire. Ces zones mémoire réservées contiennent essentiellement des données système indispensables au bon fonctionnement du ST (état des timers, horloge interne, informations sur le mode graphique en cours, etc.). Un programme utilisateur qui se plante ne peut donc pas altérer les variables système. La majeure partie des programmes Atari sont écrits en mode utilisateur. Seuls les utilitaires très spécifiques comme des débogueurs ou des moniteurs disque travaillent en mode superviseur. L'instruction POKE permet d'écrire une valeur quelconque à une adresse mémoire. Etant une instruction normale du *GFA Basic*, elle fonctionne en mode utilisateur et ne peut accéder à certaines zones de la mémoire et tout particulièrement aux 2048 premiers octets de la mémoire ST qui contiennent toutes la majeure partie des variables système du ST. L'instruction SPOKE fonctionne exactement de la même manière que POKE, mais utilise le mode superviseur. On peut donc l'utiliser pour écrire des informations à n'importe quelle adresse mémoire, y compris dans les 2048 premiers octets. Il est déconseillé de toucher au contenu de ces variables système. Toutes les modifications de l'état du ST doivent être effectuées par l'intermédiaire des fonctions BIOS et XBIOS prévues à cet effet. Sans cela, vos programmes risquent d'être dans l'impossibilité de fonctionner sur une nouvelle version de ROM. La plupart des problèmes de compatibilité entre STF et STE viennent de programmeurs qui ont négligé les spécifications de programmation Atari.
Patrick Leclercq ■

Listing 1

```
' ******************************
' *  COPIE D'UN FICHIER        *
'' *  nom1$: Fichier _ copier  *
' *  nom2$: Nom de la copie    *
' ******************************
PROCEDURE copie_fichier(nom1$,nom2$)
  LOCAL taille%
  LOCAL n%
  '
  OPEN "i",#1,nom1$   ! OUVERTURE FICHIER ORIGINE
  taille%=LOF(#1)     ! LECTURE TAILLE FICHIER
  OPEN "o",#2,nom2$   ! OUVERTURE FICHIER COPIE
  FOR n%=1 TO taille% ! DEBUT BOUCLE DE COPIE
    v%=INP(#1)        ! LECTURE D'UN OCTET
    OUT #2,v%         ! ECRITURE OCTET DANS COPIE
  NEXT n%             ! FIN BOUCLE COPIE
  CLOSE #2            ! FERMETURE FICHIER COPIE
  CLOSE #1            ! FERMETURE FICHIER ORIGINE
RETURN
```

listing 2

```
FOR i=1 TO 10
  LOCATE 1,1       ! POSITIONNEMENT AFFICHAGE
  PRINT SPACE$(5)  ! EFFACEMENT ZONE AFFICHAGE
  LOCATE 1,1       ! POSITIONNEMENT AFFICHAGE
  PRINT i          ! AFFICHAGE VARIABLE
  t=i*2
NEXT i
```

Listing 3

```
DIM t$(100)          ! TABLEAU ALPHANUMERIQUE
...                  ! REMPLISSAGE DU TABLEAU
...                  ! A PROGRAMMER VOUS-MEME
'
m$="Melnibonné"      ! CHAINE A CHERCHER
rep=-1               ! INITIALISATION VARIABLE
FOR i=1 TO 100       ! DEBUT BOUCLE DE RECHERCHE
  IF t$(i)=m$        ! TEST SI PRESENCE CHAINE
    rep=i            ! MEMORISATION POSITION CHAINE
  ENDIF              ! FIN TEST PRESENCE CHAINE
NEXT i               ! FIN BOUCLE DE RECHERCHE
'
IF rep=-1
  PRINT "Recherche infructueuse"
ELSE
  PRINT rep
ENDIF
```

Listing 4

```
m$="Melnibonné" ! CHAINE A CHERCHER
rep=-1          ! INITIALISATION VARIABLE
FOR i=1 TO 100  ! DEBUT BOUCLE DE RECHERCHE
  IF t$(i)=m$   ! TEST SI PRESENCE CHAINE
    rep=i       ! MEMORISATION POSITION CHAINE
    i=100       ! VALEUR DE FIN DE LA BOUCLE
  ENDIF         ! FIN TEST PRESENCE CHAINE
NEXT i          ! FIN BOUCLE DE RECHERCHE
```

Listing 5

```
' ******************************************
' *  CHARGEMENT D'UNE IMAGE DEGAS ELITE  *
' ******************************************
PROCEDURE charge_degas(nom$)
  LOCAL pal$
  '
  pal$=SPACE$(32)                     ! MEMOIRE PALETTE
  OPEN "I",#1,nom$                    ! OUVERTURE FICHIER
  SEEK #1,2                           ! SAUTER RESOLUTION
  BGET #1,VARPTR(pal$),32             ! CHARGEMENT PALETTE
  VOID XBIOS(6,L:VARPTR(pal$))        ! ACTIVATION PALETTE
  BGET #1,XBIOS(2),32000              ! CHARGEMENT IMAGE
  CLOSE #1                            ! FERMETURE FICHIER
RETURN
```

Listing 6

```
RESERVE 50000  ! RESERVATION MEMOIRE
SGET img1$     ! LECTURE IMAGE 1
SGET img2$     ! LECTURE IMAGE 2
END            ! FIN PROGRAMME
```

Listing 7

```
RESERVE 50000  ! RESERVATION MEMOIRE
SGET img1$     ! LECTURE IMAGE 1
SGET img2$     ! LECTURE IMAGE 2
END            ! FIN PROGRAMME
```

Listing 8

```
RESERVE 200000
DIM img$(5)
'
@charge_degas("IMAGE1.PI1")
SGET img$(1)
@charge_degas("IMAGE2.PI1")
SGET img$(2)
```

```
@charge_degas("IMAGE3.PI1")
SGET img$(3)
'
CLS            !EFFACEMENT DE L'ECRAN
VOID INP(2)    !ATTENTE PRESSION SUR CLAVIER
FOR i=1 TO 3   !DEBUT BOUCLE AFFICHAGE
   SPUT img$(i) !AFFICHAGE IMAGE I
   PAUSE 25    !ATTENTE 1/2 SECONDE
NEXT i         !FIN BOUCLE
END
```

Listing 9

```
PROCEDURE exec_travail
  DEFMOUSE 2  ! SOURIS EN FORME D'ABEILLE
  .....       ! TRAITEMENT QUELCONQUE
  DEFMOUSE 0  ! SOURIS EN FORME DE FLECHE
RETURN
```

Listing 10

```
' Ecrire dans fichier Octobre B ' (moins fiches
supprimées)
PROCEDURE ecrioct
  PUT #11,i%
RETURN
'
' Lire fichier A
PROCEDURE liroct1
  GET #10,i%
RETURN
'
PROCEDURE supnoms
  FOR i%=1 TO n%
  @liroct1
  IF LEFT$(nom$,1)="*"
    GOTO loc
  ELSE
    IF LEFT$(nom$,1)="O"  ! TEST FIN DE FICHIER
      GOSUB fermoct       ! FERMER FICHIER
    ELSE
      @ecrioct
    ENDIF
  ENDIF
  loc:
    i%=i%+1
 NEXT i%
RETURN
```

Listing 11

```
' OUVERTURE DES FICHIERS
' A ACCES DIRECTS
' ...
sortie%=0 ! CONDITION DE SORTIE BOUCLE
n%=1      ! COMPTEUR D'ELEVES DANS FICHIER A
eleve%=1  ! COMPTEUR D'ELEVES DANS FICHIER B
DO
  GET #11,n%              ! LECTURE ELEVE
  INC n%                  ! GESTION COMPTEUR FICHIER A
'
  IF LEFT$(nom$,1)="O"   ! TEST SI FIN FICHIER
    PUT #11,eleve%        !  ECRIRE FIN FICHIER
   sortie%=1              !  ACTIVER SORTIE BOUCLE
  ENDIF
 EXIT IF sortie%=1        ! TEST SI SORTIE BOUCLE
 '   IF LEFT$(nom$,1)<>"*" ! TEST SI ELEVE A ELIMINER
   PUT #11,eleve%         ! ECRIRE ELEVE FICHIER B
   INC eleve%             ! GESTION COMPTEUR FICHIER B
 ENDIF     LOOP
' FERMETURE DES FICHIERS
' A ACCES DIRECTS
' ....
```

Listing 12

```
' ********************
' *  A NE PAS FAIRE  *
' *  AVEC GOTO       *
' ********************
IF LEFT$(nom$,1)="*"
  GOTO loc:
ELSE
  IF LEFT$(nom$,1)="O"
  ...
  ENDIF
ENDIF
loc:
' ***************
' *  A FAIRE    *
' *  SANS GOTO  *
' ***************
IF LEFT$(nom$,1)<>"*"
  IF LEFT$(nom$,1)="O"
  ...
  ENDIF
ENDIF
```

PREMIERS PAS EN OMIKRON
Les fichiers (III)

Explorons toujours plus avant le domaine des fichiers en Omikron, et voyons si l'on peut trouver une solution qui allierait la rapidité du fichier à accès direct et la souplesse du fichier séquentiel.

Nous avons vu que le fichier séquentiel était particulièrement simple à programmer mais peu exploitable pour les gros fichiers compte tenu de sa lenteur, un enregistrement ne pouvant être lu directement. A l'inverse, le fichier à accès direct comble les inconvénients du fichier séquentiel en terme de rapidité, mais suppose que l'on se souvienne des numéros d'enregistrement. En fait, l'inconvénient majeur du fichier à accès direct c'est le numéro d'enregistrement. Par conséquent, il faut imaginer un système pour que ces indices soient mémorisés automatiquement. C'est le fichier séquentiel indexé.

Qu'est-ce qu'un index?

Lorsque l'on recherche une fiche unique, on base sa recherche sur un minimum d'informations. Prenons un exemple simple. Supposons que vous créez un fichier client comportant les champs suivants: nom, prénom, adresse, téléphone, etc. Si vous désirez obtenir des renseignements sur le client Dupont, vous entrez son nom et le programme vous affiche les renseignements le concernant. Cet exemple vous montre bien que, pour ce type de recherche, vous êtes obligé de connaître au moins le nom de la personne. Il est donc judicieux d'indexer votre fichier sur le champ «Nom». Cela veut dire en clair que vous allez créer, en plus de votre fichier principal, un fichier secondaire, un index, dans lequel figureront les noms de votre fichier principal et les numéros d'enregistrement correspondants. Cet index sera un fichier séquentiel à deux champs: le nom et l'indice d'enregistrement. Par cette méthode, en cherchant la fiche de Dupont, le programme regardera d'abord dans l'index si le nom existe, et s'il le trouve, en obtiendra le numéro d'enregistrement pour obtenir le reste des renseignements dans le fichier principal.

La question qui peut se poser est: comment cela peut-il aller plus vite, puisque cela oblige à consulter deux fichiers au lieu d'un? La lecture séquentielle d'un fichier de un ou deux champs in: toujours plus vite que la lecture séquentielle d'un gros fichier à accès direct comportant une dizaine de champs (voire plus si vous décidez de faire un fichier vraiment complet). Autrement dit, cela prend plus de temps de scruter directement le fichier principal, que de chercher l'indice d'enregistrement dans l'index, puis de s'en servir pour lire la fiche convoitée dans le fichier principal.

Comment gérer le fichier d'index?

Il existe plusieurs méthodes qui dépendent de vos besoins personnels et de votre configuration matérielle:

- il est tout à fait concevable de créer plusieurs fichiers d'index afin d'étendre la souplesse de recherche. Exemple: un index «Nom» et un index «Téléphone». Si vous possédez un disque dur, cela ne pose pas trop de problèmes de multiplier les fichiers,
- le fichier d'index peut très bien ne contenir qu'un seul champ: l'index lui-même. Dans ce cas, l'indice d'enregistrement est implicitement le numéro d'ordre dans lequel la fiche a été enregistrée. Cependant, cette méthode empêche de se servir de l'index comme d'un fichier à part entière. En effet, il peut parfois être utile de trier l'index (donc de modifier l'ordre de départ) et de le lister,
- si vous estimez connaître le nombre maximum de fiches que vous atteindrez jamais, vous pouvez toujours dimensionner un tableau dans lequel vous chargerez l'index en début de programme. De ce fait, la recherche de l'indice se fera en mémoire, ce qui est pratiquement instantané. Cette méthode suppose que vous gérez le tableau en même temps que le fichier d'index en cas d'ajout de fiches.

Quelques exemples

Commençons par adapter le listing d'exploitation du fichier à accès direct paru dans Atari Magazine n°36, et ajoutons-lui une gestion d'index.

```
E$="X"+ CHR$(27)
OPEN "R",1,"A:\ESSAI.DAT",48
OPEN "O",2,"A:\INDEX.DAT"
FIELD 1,20 AS N_$,20 AS P_$,2 AS A_$,6 AS S_$
CLS :Ind%= LOF(1)+1
```

```
PRINT "Arrêt de la saisie : [ESC]."
REPEAT
 PRINT @(2,2);"Nombre d'enreg. :"; LOF(1)
 N$="":P$="":A$="":S$=""
 INPUT @(5,5);"Nom : ";N$ USING "aU"+E$,R,20
 IF (R AND $FF)=27 THEN EXIT
 WRITE #2,N$,Ind%
 INPUT @(6,2);"Prénom : ";P$ USING "a+-",R,20
 INPUT @(7,5);"Age : ";A$ USING "0",R,3
 INPUT @(8,3);"Solde : ";S$ USING "0+.+,C,.",R,10
 RSET N_$=N$: RSET P_$=P$
 A_$= MKI$( VAL(A$)):S_$= MKS$( VAL(S$))
 PUT 1,Ind%
 Ind%=Ind%+1
UNTIL N$=""
CLOSE 2
PRINT @(11,2);"Une touche pour lire le fichier..."
REPEAT UNTIL LEN( INKEY$ )
'
CLS
PRINT "Arrêt de la lecture : [ESC]."
REPEAT
 N$=""
 INPUT @(2,1);"Nom : ";N$ USING "aU"+E$,R,20
 IF (R AND $FF)=27 THEN EXIT
 OPEN "I",2,"A:\INDEX.DAT"
 Flag%=0
 WHILE NOT EOF(2)
   INPUT #2,Nom$,Ind%
   IF N$=Nom$ THEN Flag%=-1: EXIT
 WEND
 IF Flag% THEN
   GET 1,Ind%
   PRINT @(4,1); CHR$(27);"K";
   PRINT N_$,P_$, CVI(A_$), CVS(S_$)
 ELSE
   PRINT @(0,0); CHR$(7);
 ENDIF
 CLOSE 2
UNTIL N$=""
'
CLS : CLOSE : END
```

On s'aperçoit que la recherche est beaucoup plus rapide. Mais pour en avoir le cœur net, reprenons le test comparatif de notre première rubrique (Atari Magazine n°35) consacrée aux fichiers qui consistait à écrire un fichier de mille enregistrements et de rechercher le dernier. Rappelons qu'avec un fichier séquentiel, il nous a fallu, pour lire la dernière fiche, plus de douze secondes sur disquette et plus de cinq secondes sur disque dur.

```
CLS : PRINT CHR$(27);"f"
PRINT "Insérez une disquette A:"
PRINT "et appuyez sur une touche..."
REPEAT UNTIL LEN( INKEY$ )
OPEN "O",1,"A:\INDEX.DAT"
OPEN "R",2,"A:\ESSAI.DAT",42
FIELD 2,20 AS N_$,20 AS P_$,2 AS S_$
CLS : PRINT : PRINT "Ecriture :"
FOR I%=1 TO 999
  N$="":P$= CHR$(65+ RND(24))
  FOR U%=5 TO RND(5)+10
    N$=N$+ CHR$(65+ RND(24))
    P$=P$+ CHR$(97+ RND(24))
  NEXT U%
  S%= RND(10000)
  WRITE #1,N$
  LSET N_$=N$: LSET P_$=P$:S_$= MKI$(S%)
  PUT 2,I%: PRINT @(1,10);I%
NEXT I%
WRITE #1,"ATARI"
LSET N_$="ATARI": LSET P_$="Magazine"
S_$= MKI$(1992)
PUT 2,1000
CLOSE 1
PRINT : PRINT "OK ! Appuyez sur une touche"
PRINT "pour commencer la recherche..."
REPEAT UNTIL LEN( INKEY$ )
Ti= TIMER
OPEN "I",1,"A:\INDEX.DAT"
I%=1
WHILE NOT EOF(1)
  INPUT #1,N$
  IF N$="ATARI" THEN EXIT TO Ok
  I%=I%+1
WEND
PRINT : PRINT "Non trouvée ! Fin de fichier..."
CLOSE : END
-Ok
GET 2,I%
PRINT : PRINT "Fiche : ";N_$;P_$; CVI(S_$)
PRINT "trouvée en";( TIMER -Ti)/200;" sec."
PRINT "Taille du fichier :"; LOF(2)*42;" octets."
CLOSE
```

Cette fois, cela devient honorable puisque sur disquette la dernière fiche est trouvée en environ six secondes, et sur disque dur en un peu plus de deux secondes. Nous avons divisé le temps de recherche par deux. Fort de cet avantage, nous allons encore optimiser nos temps d'accès en cherchant l'index en mémoire...

```
CLS : PRINT CHR$(27);"f"
DIM T$(1000)
Ti= TIMER
PRINT "Chargement de l'index...";
OPEN "I",1,"A:\INDEX.DAT"
I%=1
WHILE NOT EOF(1)
 INPUT #1,T$(I%)
 I%=I%+1
```

```
WEND
CLOSE 1
PRINT " en";( TIMER -Ti)/200;" sec."
OPEN "R",2,"A:\ESSAI.DAT",42
FIELD 2,20 AS N_$,20 AS P_$,2 AS S_$
PRINT : PRINT "OK ! Appuyez sur une touche"
PRINT "pour commencer la recherche..."
REPEAT UNTIL LEN( INKEY$ )
Ti= TIMER
FOR I%=1 TO 1000
  IF T$(I%)="ATARI" THEN EXIT TO Ok
NEXT I%
PRINT : PRINT "Non trouvée ! Fin de fichier..."
CLOSE : END
-Ok
GET 2,I%
PRINT : PRINT "Fiche : ";N_$;P_$; CVI(S_$)
PRINT "trouvée en";( TIMER -Ti)/200;" sec."
PRINT "Taille du fichier :"; LOF(2)*42;" octets."
CLOSE
```

A ce niveau, cela devient très intéressant puisque sur disquette le dernier enregistrement est lu en moins de deux secondes, et sur disque dur en mois de cinq dixièmes de seconde. C'est vrai qu'il faut compter le temps de chargement de l'index au début du programme, mais celui-ci est chargé une fois pour toutes, et son temps de chargement se noie très largement dans celui du programme.

Les mauvaises surprises

Essayons maintenant de voir quels peuvent être les messages d'erreur liés aux fichiers. Commençons par les plus simples à corriger:

Bad file number: un numéro d'identification de fichier n'est pas compris entre 1 et 16.

File already open: vous avez cherché à ouvrir un fichier alors qu'il était déjà ouvert, c'est-à-dire qu'il n'a pas été fermé auparavant par l'instruction CLOSE.

File not open: vus avez cherché à écrire ou lire des données dans un fichier qui n'a pas été ouvert par l'instruction OPEN.

Field overflow: la taille des données contenues dans les champs d'un fichier à accès direct dépasse la capacité mémoire du tampon de l'instruction FIELD. Votre fichier doit être énorme! Dans ce cas, utilisez plusieurs fichiers parallèles.

Bad record number: le programme a tenté de lire dans un fichier à accès direct un enregistrement dont le numéro dépasse la capacité du GEMDOS.

Bad file name: le nom du fichier que vous avez tenté d'ouvrir possède des caractères interdits (codes VT-52 par exemple).

File not found ou *Path not found*: le fichier que vous avez tenté d'ouvrir n'existe pas, ou bien le chemin d'accès précisé n'existe pas.

Passons aux messages d'erreurs plus difficiles à interpréter:

Input past end: ce message d'erreur surgit lorsque le programme tente de lire une donnée au-delà de la fin d'un fichier séquentiel. Bien sûr, cela peut provenir tout simplement d'une étourderie comme oublier de tester la fin du fichier. En général, on procède de l'une des deux manières suivantes:

```
WHILE NOT EOF(N)
  Lecture des données...
WEND
```

ou

```
REPEAT
  IF EOF(N) THEN EXIT
  Lecture des données...
UNTIL 0
```

Attention : la boucle REPEAT ... (lecture) ... UNTIL EOF(N) n'est pas adaptée, car le test de la condition de fin de fichier est effectué après la lecture. Par conséquent, le message d'erreur peut apparaître.

Mais ce n'est pas aussi simple qu'il y paraît. Cette erreur est aussi engendrée par d'autres phénomènes:

- il peut s'agir d'une mauvaise lecture par rapport à la méthode d'écriture employée. Exemple: écrire des enregistrements sur un champ et vouloir les lire sur plusieurs,
- il peut s'agir de caractères spéciaux tels que les codes ASCII 13, 44 ou 26 sauvés par erreur dans un enregistrement et qui perturbent la lecture des données.

Bad file mode: le type d'ouverture du fichier ne correspond pas à l'action entreprise. Par exemple: tenter d'écrire dans un fichier séquentiel ouvert en lecture par OPEN "I". Il s'agit en général d'une confusion dans les numéros d'identification des fichiers, lorsque plusieurs fichiers sont ouverts simultanément. Utilisez l'option "LIST TOKEN..." de l'éditeur afin de lister les lignes où se trouvent l'instruction OPEN.

Il existe un troisième type de fichier, le fichier «Utilisateur» ouvert par OPEN "U", très proche de son modèle fondamental du GEMDOS "Fopen".

Pierre-Jean Goulier ■

GESTION DOCUMENTAIRE
Faites-le vous même

La meilleur manière d'apprendre à programmer, c'est d'écrire un programme complet. C'est pourquoi nous vous donnons aujourd'hui la description d'un programme intéressant que vous pourrez réaliser par vous-mêmes.

Logiciel de gestion documentaire?

Un logiciel de gestion documentaire est un programme qui permet de rechercher des documents ou des articles de journaux d'après un certain nombre de critères.
Par exemple, un logiciel de gestion documentaire est capable d'indiquer tous les articles d'Atari Magazine traitant des traitements de texte et du graphisme. C'est un programme pouvant être très utile pour le possesseur d'une grande quantité de journaux. Une bibliothèque peut l'utiliser pour gérer ces revues et ces différents livres. Dans ce dernier cas, il peut être lié avec le système d'abonnement.

Définition d'une fiche article type

Les fiches contenant la description des articles peuvent contenir un titre, une référence de journal, le nom de l'auteur, une date de parution, un texte de présentation et les critères de recherche. Si la bibliothèque est importante, vous pouvez y rajouter le numéro de l'armoire ou du rayon où est stocké le journal. Les critères de recherches sont des mots résumant le contenu du programme. Par exemple, un article sur l'affichage de sprites en Gfa Basic parus dans Atari Magazine n°30 doit avoir les critères PROGRAMMATION, GFA BASIC, GRAPHSIME et SPRITES. Rien ne vous empêche de rajouter d'autres informations sur les articles, comme leurs numéros de pages, une note ou une appréciation personnelle, etc..
La manière dont les données seront stockées dans la mémoire dépend du langage de programmation, mais on peut remarquer que certaines données sont répétitives, comme les titres des journaux et le nom des auteurs. Plutôt que de les garder à chaque fois en mémoire en perdant pas mal de place mémoire, il est préférable de leurs donner un numéro de référence correspondant à une liste de noms conservé quelque part en mémoire. Par exemple, Atari Magazine pourrait avoir le numéro 1, Cassus Belli le numéro 2, SVM le numéro 3, etc.. Avec ce système, le codage du journal et de l'auteur d'un article ne prend plus que 3 ou 4 octets à la place d'une cinquantaine. L'économie de mémoire est appréciable!

Etudes des fonctionnalités

L'utilisateur d'un logiciel de gestion documentaire simple doit pouvoir ajouter des articles à la base de données et rechercher un article à partir d'une liste de critères. Le résultat de la recherche doit être affiché sur l'écran ou imprimé. Toutes ces opérations doivent se faire simplement, avec une interface fonctionnant à la souris.

Création d'une fiche article

Pour créer une nouvelle fiche article, l'utilisateur doit saisir tous les renseignements qu'il connaît sur l'article. Le système le plus simple est d'utiliser l'instruction INPUT pour saisir individuellement chaque information. C'est le procédé le plus simple à programmer, mais pas le plus simple à utiliser! Certaines informations peuvent déjà être connus du programme, comme le titre du journal, le nom de l'auteur et certains critères de recherche.
Dans ce cas, l'utilisateur doit avoir la possibilité de sélectionner l'une des informations déjà présente en mémoire sans la re-saisir.
L'image accompagnant cet article vous montre une disposition possible pour l'écran de saisie.
La saisie d'un élément de l'article se fait en cliquant dessus. Par exemple, pour entrer le titre de l'article, il faut cliquer sur la zone TITRE, puis taper directement le texte. En cliquant sur un nom de journal déjà présent sur l'écran, on modifie le nom du journal de l'article en cours de saisie. Cela fonctionne de la même manière pour les noms des auteurs et pour les critères de recherches. Lorsque l'utilisateur saisie un titre de journal, un nom d'auteur ou un critère de recherche inconnu du programme, celui-ci l'ajoute à ses listes et l'affiche sur l'écran. Les flèches hauts et bas des différentes fenêtres permettent d'afficher la totalité des informations possédés par le programme.
Il est en effet peu probable que la base de donnée ne contiennent que 4 magazines différents ou seulement 4 auteurs, sans parler des nombreux critères de recherches possi

PROGRAMMATION

LNADOCS	contient une documentation complète sur le langage LNA de la société Lexiel	21624	arc
PBAS45	v. 4.5 du PBasic, interpréteur Basic qui possède un jeu d'instructions et gère le graphisme du Portfolio (PLOT, LINE, etc.)	164273	arc
PRGCALC	calculatrice pour programmeur permettant de travailler en binaire, octal (base 8), décimal et hexadécimal	11531	arc
SCI	interpréteur C qui permet de tester un programme C de la même manière qu'un programme Basic	127293	arc
TBASIC	mini basic limité à 2795 octets qui ne comprend que les ordres donnés en majuscules et ne peut gérer que 26 variables	37903	arc
UNIFORTH	adaptation du Forth Uniforth Professionnal. Il supporte le standard Forth-83	107190	arc

GRAPHISME

PGCMP1	contient le programme PGCOMP permettant de convertir des fichiers PGF en fichier PGC	8240	arc
PGSPEC	explications complètes sur la structure du format graphique PGC (Portfolio Graphic Compressed)	3293	arc
PGSH21	v. 2.1 du programme PGSHOW. Cet utilitaire permet de visualiser des images sur l'écran du Portfolio et reconnaît les formats graphiques PGF et PGC	8027	arc
SKETCH	programme de dessin du type «écran magique». Les curseurs permettent de tracer des traits dans n'importe quelle direction	6246	arc
SPRITE	démonstration des capacités graphiques du Portfolio. Une trentaine de petits sprites se déplacent rapidement et rebondissent sur les bords de l'écran!!!	1488	n.c

COMMUNICATION

EMMASP	utilitaire de capture de messages sur le courrier électronique US	24648	arc
PRTFTERM	programme de terminal ASCII qui permet la communication série et de transférer des programmes sur PC plus rapidement qu'avec l'interface parallèle	15198	arc
TMXM.DOC	documentation anglaise du programme TMXM.COM	2737	n.c
TMXM2.DOC	documentation anglaise du programme TMXM2.COM	4919	n.c
TMXM2	permet de disposer des protocoles Xmodem et Ymodem sur Portfolio	2432	n.c
TMXM	permet de disposer des protocoles Xmodem et Ymodem sur le Portfolio	2048	n.c

UTILITAIRES DOS

FM	SHELL texte qui simplifie l'utilisation du Portfolio	7680	n.c
PSTAT	utilitaire qui affiche différentes informations sur l'état du Portfolio (état du clavier, de l'écran, type de Beecard, horloge, etc.)	2441	arc
PTOOL	ensemble de 9 utilitaires. Le programme Portdisk donne des informations sur les disques	27023	arc
PUTILS	utilitaires pour Portfolio. Le programme SDIR affiche la liste des fichiers d'un disque	9538	arc
SM	utilitaire DOS permettant de choisir le mode écran: Static, Tracked ou normal	128	n.c
UPDATE	utilitaire de correction des bugs du Portfolio (date de mise en service: 17/12/90)	1664	n.c
UPDATE14	programme de mise à jour qui permet de corriger les quelques bugs mineurs du système d'exploitation. Il doit être exécuté à chaque réinitialisation du Portfolio ou lancé à partir du fichier AUTOEXEC.BAT	1536	n.c

DIVERS

CBASE	permet de saisir des informations selon un formulaire défini à l'avance (utile pour une enquête ou un sondage)	26466	arc
PORTFO	gère sur ST les fichiers d'adresses et de numéros de téléphone au format du Portfolio	33536	arc
PTONE	utilitaire musicale transformant le Portfolio en mini synthétiseur. Logiciel prévu pour un clavier QWERTY	7807	n.c

Définition des abréviations

n.c: fichier non compacté. Utilisable tel quel.
a.d: fichier auto-décompactable. Double cliquez dessus pour le décompacter.
arc: fichier compacté avec ARC. Utilisez ARCX.TTP pour le décompacter.
ar: fichier compacté avec Archive. Utilisez ARCHIVE.TTP pour le décompacter.
c14: fichier compacté avec Compi14. A décompacter avec le programmme COMPI14.PRG.
deg: image au format Degas compressé .PC?
tny: image au format Tiny.